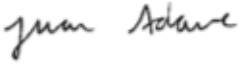
Real-time Visual Flow Algorithms for Robotic Applications

Juan David Adarve Bermudez

A thesis submitted for the degree of
Doctor of Philosophy
The Australian National University

November 2017

Except where otherwise indicated, this thesis is my own original work.



Juan David Adarve Bermudez
6 November 2017

To my beloved parents, Consuelo and Omar.

Acknowledgments

This thesis has been the work of many years and several (many) steps in the wrong direction. Throughout the journey, I have learned the meaning of doing quality research and to think about some fundamental questions currently important in robotics. None of this would have been possible without the guidance of Robert Mahony, my mentor and supervisor. Our regular discussions on how to best realize our algorithms have shaped my scientific thinking. I am very happy and grateful for being part of his school of thought. A special thanks to the members of my thesis committee members, Richard Hartley, Eric McCreath and David Austin for their advice throughout my PhD candidature.

To my fellow PhD students, Moses Bangura, Christian Rodriguez, Yi Zhou, Xiaolei Hou, Geoff Stacey and Alireza Khosravian, a huge thanks for their support along the PhD journey and for making my time at ANU very enjoyable. Also, many thanks to Alex Martin for his help building the required hardware and assistance running the experiments. To all my friends: Johnson, Edwin, Yessica, Belinda, Giulia, Johnny and Jhonatan, I value you all for being there when I needed it most.

Last but not least, I am deeply grateful to my family and beloved parents Consuelo and Omar for their love, support and sacrifices. To my mom, I am forever grateful for your love and advice, and my dad, although he passed away long ago, inspired me to follow the engineering path.

This work was supported initially by the Australian Research Council and MadJInnovation through the Linkage grant LP11020076 and later by the Australian Research Council through the "Australian Centre of Excellence for Robotic Vision" CE140100016.

Abstract

Vision offers important sensor cues to modern robotic platforms. Applications such as control of aerial vehicles, visual servoing, simultaneous localization and mapping, navigation and more recently, learning, are examples where visual information is fundamental to accomplish tasks. However, the use of computer vision algorithms carries the computational cost of extracting useful information from the stream of raw pixel data. The most sophisticated algorithms use complex mathematical formulations leading typically to computationally expensive, and consequently, slow implementations. Even with modern computing resources, high-speed and high-resolution video feed can only be used for basic image processing operations. For a vision algorithm to be integrated on a robotic system, the output of the algorithm should be provided in real time, that is, at least at the same frequency as the control logic of the robot. With robotic vehicles becoming more dynamic and ubiquitous, this places higher requirements to the vision processing pipeline.

This thesis addresses the problem of estimating dense visual flow information in real time. The contributions of this work are threefold. First, it introduces a new filtering algorithm for the estimation of dense optical flow at frame rates as fast as 800 Hz for 640×480 image resolution. The algorithm follows a update-prediction architecture to estimate dense optical flow fields incrementally over time. A fundamental component of the algorithm is the modeling of the spatio-temporal evolution of the optical flow field by means of partial differential equations. Numerical predictors can implement such PDEs to propagate current estimation of flow forward in time. Experimental validation of the algorithm is provided using high-speed ground truth image dataset as well as real-life video data at 300 Hz.

The second contribution is a new type of visual flow named *structure flow*. Mathematically, structure flow is the three-dimensional scene flow scaled by the inverse depth at each pixel in the image. Intuitively, it is the complete velocity field associated with image motion, including both optical flow and scale-change or apparent divergence of the image. Analogously to optic flow, structure flow provides a robotic vehicle with perception of the motion of the environment as seen by the camera. However, structure flow encodes the full 3D image motion of the scene whereas optic flow only encodes the component on the image plane. An algorithm to estimate structure flow from image and depth measurements is proposed based on the same filtering idea used to estimate optical flow.

The final contribution is the *spherepix* data structure for processing spherical images. This data structure is the numerical back-end used for the real-time implementation of the structure flow filter. It consists of a set of overlapping patches covering the surface of the sphere. Each individual patch approximately holds properties such as orthogonality and equidistance of points, thus allowing efficient implementations of low-level classical 2D convolution based image processing routines such as Gaussian filters and numerical derivatives.

These algorithms are implemented on GPU hardware and can be integrated to future Robotic Embedded Vision systems to provide fast visual information to robotic vehicles.

Contents

List of Figures

List of Tables

Chapter 1

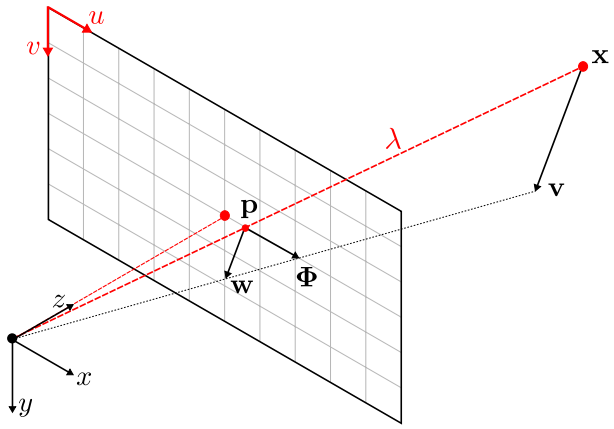
Introduction

> Genius is one percent inspiration and ninety-nine percent perspiration.
>
> Thomas Alva Edison

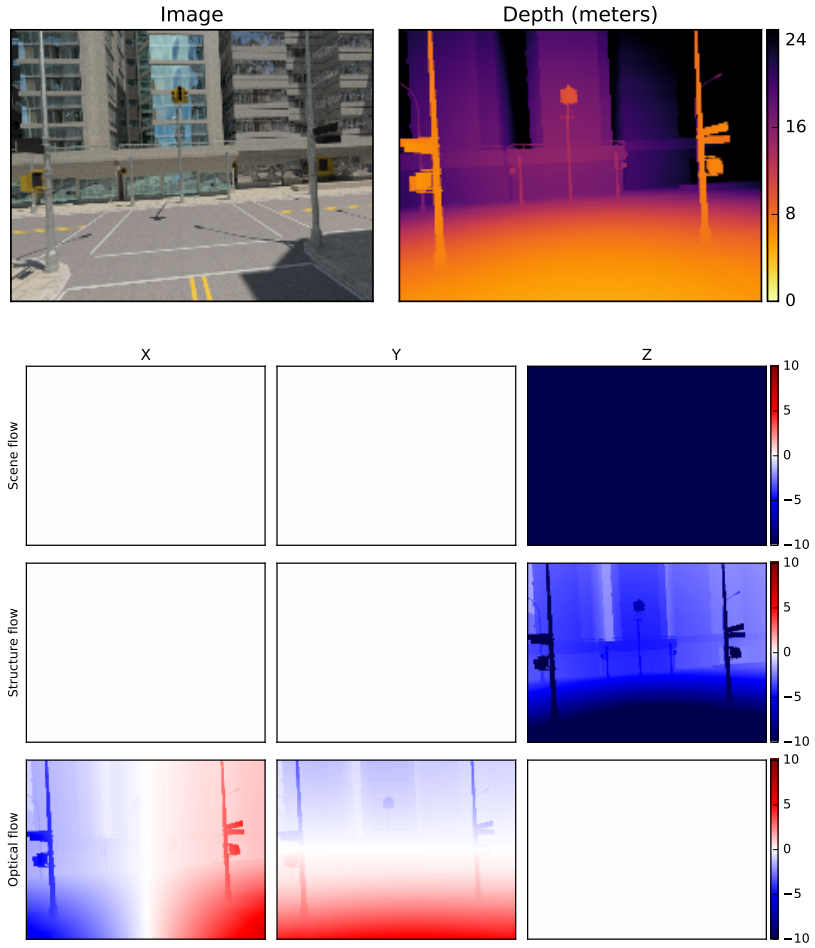
Vision offers rich sensor information to robotic vehicles interacting in complex dynamic environments. As robots are increasingly deployed in unconstrained dynamic environments, the requirements of the visual system become more demanding in terms of accuracy and response time. Thanks to the expansion of the mobile phone industry, image sensors and embedded processors have experienced an increase in compute performance as well as a reduction in price, making the use of such technologies cost effective within the robotics industry. However, the extraction of meaningful information from a vision sensor is a complicated and computationally intensive task.

The focus of this thesis is on the design and implementation of dense real-time *visual flow* algorithms. A visual flow field is a vector field on the camera's image surface that provides motion information for each pixel in the image. There are at least three types of visual flows, illustrated in Figure 1.1a, that can be extracted from image sequences: *scene flow*, *optical flow* and the novel *structure flow* that I introduce in Chapter 4. The *scene flow*, so called by Vedula et al. [1999], is the three-dimensional motion field of points in the world relative to the camera. That is, the Euclidean 3D velocity between the closest object in the scene at a given pixel location and the robot's camera. *Optical flow* is the projection of the scene flow onto the image plane of the camera. It describes the velocity of each pixel in the image plane induced by the motion of the robot and the environment Barron [1994]. *Structure flow*, introduced in this thesis Adarve and Mahony [2016b], sits in between scene and optical flow. Mathematically, structure flow is the three-dimensional scene flow scaled by the inverse depth at each pixel in the image. Intuitively, it is the complete velocity field associated with image motion, including both optical flow and scale-change or apparent divergence of the image. Analogously to optic flow, structure flow provides a robotic vehicle with perception of the motion of the environment as seen by the camera. However, structure flow encodes the full 3D image motion of the scene whereas optic flow only encodes the tangential image motion.

From an algorithm complexity perspective, optical flow is the easiest to compute from image data among the three types of visual flow. Estimation of dense optical flow can be seen as a dense registration of two images separated in time Lucas and Kanade [1981]. For



(a) Scene $\mathbf{v} \in \mathbb{R}^3$, structure $\mathbf{w} \in \mathbb{R}^3$ and optical flow $\mathbf{\Phi} \in \mathbb{R}^2$ vectors for a point $\mathbf{x}$ in the scene. Point $\mathbf{x}$ moves with a relative velocity (scene flow) $\mathbf{v}$ with respect to the camera body fixed frame, and projects to pixel $\mathbf{p}$ on the image plane. The optical flow $\mathbf{\Phi}$ at $\mathbf{p}$ is a two-dimensional projection of scene flow on the image plane. The structure flow vector $\mathbf{w}$ is the scaling of the scene flow by the inverse of the distance λ to $\mathbf{x}$.



(b) Scene, structure and optical flow fields for a perspective camera mounted on a forward moving vehicle at 10 $^m/_s$. The simulated perspective camera runs at 100 Hz.

Figure 1.1: Scene, optical and structure flow fields.

each pixel in the first image, a displacement vector is computed such that the brightness value in the second image matches. Consequently, computation of optical flow is purely a data matching process, and it does not depend directly on the underlying scene or camera geometry. Algorithms for estimating optical flow can be counted by the hundreds and they have been summarized over the last two decades in the survey papers of Barron [1994] and Baker et al. [2011]. Optical flow has been widely used in robotic systems. Application examples include: visual servoing Hamel and Mahony [2002], vehicle landing on a moving platform Herisse et al. [2012], height regulation Ruffier and Franceschini [2005] and obstacle avoidance Srinivasan [2011a].

Computation of Scene flow is less studied than optical flow. State of the art algorithms using two-pair stereo-images can be found in the Kitti scene flow dataset of Menze and Geiger [2015]. Other approaches such as those using RGB-D sensors (e.g., the Microsoft Kinect) have also been studied Hadfield and Bowden [2011], Herbst et al. [2013]. Real-time scene flow algorithms are relatively slow compared to optical flow, running between 20-30 Hz Rabe et al. [2010], Wedel et al. [2011]. Runtime performance of modern two-pair stereo based approaches are reported in the Kitti scene flow dataset[1].

Figure 1.1b illustrates the scene, structure and optical fields for a vehicle moving at 10 $^{m}/_{s}$ in collision course with a building located approximately at 15 meters. The image and depth map are from the visual odometry dataset of Zhang et al. [2016]. The simulated perspective camera (with the Z axis matching the camera focal axis) runs at 100 Hz and all flow fields are expressed in pixel units. Since the simulated scene is static, the calculated scene flow is equal for all pixels in the image and is equal to the negative of the camera velocity. As the vehicle moves along the focal axis of the camera, the xy component of the scene flow are zero. Scaling the scene flow by the inverse of the depth field, we obtain the structure flow field. The structure flow distinguishes between objects close to or far away from the camera since the relative angular divergence of closer objects is larger. That is, they are growing in the image more quickly than distant objects. Last row of Figure 1.1b is the optical flow field on the image plane of the camera. Since optical flow is the projection of the scene flow onto the image plane, the z component of the optical flow is zero. That is, no motion along the focal axis of the camera. In fact, the optical flow is a divergent vector field with the focus of expansion located in the center of the image; the direction of motion. Notice that, although the vehicle is moving quickly, the optical flow in the central region of the image is small, and it is difficult to evaluate the time to contact before the vehicle collides with the building.

Current algorithms developed within the Computer Vision community aim at improving the accuracy of the estimated optical flow fields. This trend can be observed in the latest results of standard optical flow benchmark datasets Baker et al. [2011]; Butler et al. [2012]; Geiger et al. [2013]. At the same time, the complexity of the top performing algorithms has increased and thus their computational demands, as reported in the benchmarks. While these algorithms can be used in applications where runtime performance is not a critical constraint, their application on real-time vision pipelines, as those required by robotic platforms, is questionable.

An underlying requirement of any visual flow algorithm in a robotic system is the capability

[1] http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php

to perform computations in real-time. If the output of the algorithm is to be used as direct sensor feedback by the controller, 'real-time' means at least as fast as the frequency at which the control system works and preferably 5-10 times faster. For a typical aerial vehicle, this means the visual system needs to operate at frequencies in the order of 200-500 Hz Bangura et al. [2015]. Moreover, due to power and weight limitations of the vehicle, computations have to be carried out on embedded compute platforms with limited computational resources. As a consequence, typical robotic systems use rather simple algorithms, such as Srinivasan [1994] and Bouguet [2001] for computing optical flow, compared to the state of the art in Computer Vision.

Algorithms such as those mentioned above were designed using a two-frame approach in which the algorithm gives a dense output for each pair of frames. While this is the typical approach for computing optical flow, it does not match the intrinsic properties of a robotic vision system. Those are:

- *High-speed video capture*: Robotic vision systems can take advantage of high-speed image sensors to sample the evolution of the environment at high frame rates. As the environment is sampled faster, the relative change between any two consecutive frames gets smaller. In the limit, with a camera working at multiple hundreds of Hertz, the difference between any two images would be expected to be at the sub-pixel level.

- *Additional sensor modalities*: A typical robotic vehicle contains several sensors, such as inertial measurement units and laser scanners, that can be integrated into the visual system. Information provided by these sensors can be fused within the vision algorithms to create a robust and coherent representation of the environment surrounding the robot.

A key challenge of using high-speed image sensors is processing the stream of data in real-time. As an example, the high-speed camera used in the experiments of this thesis provides 1016×544 images with 256 brightness levels per pixel at 300 Hz. This is equivalent to 158 Megabytes of image data to be processed every second. This resolution and frame rate places a heavy load on the available computational resources such as USB3 and RAM bandwidth as well as the compute resources of typical GPU hardware.

Considering the properties mentioned above, I proposed the term *Robotic Embedded Vision* (REV) system to describe an electronic system and a set of algorithms to offer visual sensing capabilities to a robotic vehicle. A REV device contains vision sensors connected directly to processing elements (CPU, GPU or FPGA) as well as extra sensors, such as IMU or GPS, to provide additional information to the algorithms. Vision algorithms are executed inside the REV device, reducing the latency between image acquisition and processing. The output of the vision algorithms (optical flow, feature points, etc) is then used according to the robot's current task, and only information required to perform such task is transmitted to the robot CPU. An application example is the transmission of velocity commands for an aerial vehicle according to the perceived optical flow. Notice that only the result of the algorithm is transmitted out of the device. This can potentially reduce the required bandwidth of the communication channel between the REV system and the robot CPU.

The objective of this thesis is the development of visual flow algorithms, optical and structure flow, specifically designed for REV systems. The design process of these algorithms takes

into consideration the properties of a robotic vision system mentioned above to create real-time algorithm formulations and implementations. Both optical and structure flow algorithms follow a filtering approach for estimating the underlying visual flow. The filters follow the standard prediction-update approach to incrementally build a dense estimate of flow using new sensor measurements and predictions made using the old state.

There are two advantages of using a filtering approach. First, thanks to the incremental nature of the algorithm, the output of the algorithm is temporally consistent. The state estimate at time $k+1$ is equal to the old estimate at time k plus some innovation considering new measurements. Second, a filtering approach can reduce the amount of computations required at each time step. Instead of computing highly accurate and dense flow fields using two images, that is, following the standard approach, an incremental algorithm does partial computations at every time step and adds new information to the state. Dense and accurate state estimates are reached over time.

An important contribution of this thesis is the use of partial differential equations (PDE) to model the spatio-temporal evolution of the optical and structure flow fields on the image surface. Efficient numerical methods that match the massive parallel compute power of GPU and FPGA platforms are developed to solve these PDEs.

For each visual flow algorithm, experimental validation is provided using both ground-truth data simulating a high-speed camera mounted on a mobile vehicle. Additionally, results on real-life videos captured are provided to validate the algorithms; for optical flow, a 300 Hz high-speed monocular camera is used, while a 60 Hz stereo camera array is used to test the structure flow algorithm. All the algorithms were implemented and tested on a Nvidia GTX 780 Desktop GPU card and partially tested on a embedded Nvidia Tegra K1 System on Chip.

1.1 Contributions

The following are the contribution of this thesis towards the design and development of real-time visual flow algorithms for robotic applications.

- **Optical flow filter:** A filtering algorithm for the computation of dense optical flow in real-time is proposed. The algorithm uses a predictor-update approach for the incremental estimation of optical flow. The prediction part is the first one to introduce the PDE approach for modeling the spatio-temporal evolution of the flow field on the image plane. A GPU implementation of the algorithm is developed and it is currently available as an open-source software. The algorithm can run at frame rates of more than 300 Hz on commodity Desktop GPU hardware.

- **Structure flow filter:** The structure flow is defined geometrically as the three-dimensional scene flow scaled by the inverse depth of the scene. Intuitively, the structure flow models the evolution of the image, structure and velocity of the environment as seen from the camera, Partial differential equations to model the spatio-temporal evolution of the structure flow on spherical camera geometry are proposed. These equations are used to design a filtering algorithm for the estimation of structure flow in real-time using image and depth measurements. A GPU implementation of the algorithm is developed and can run at frame rates up to 200 Hz on 1 Mpix images.

- **Spherepix data structure for spherical image processing:** The Spherepix data structure is a discretization of the unit-sphere on which low-level image operations such as Gaussian filtering and gradient computation can be efficiently implemented. Spherepix provides the numerical layer on which the structure flow filter algorithm is implemented.

1.2 Publications

Published

1. Adarve, JD., Li, W., Mahony, R. and Austin, D., *Towards an Efficient and Robust Optic Flow Algorithm for Robotic Applications*, Australasian Conference on Robotics and Automation. 1-9, 2012.

2. Adarve, JD., Austin, D. and Mahony, R., *A Filtering Approach for Computation of Real-Time Dense Optical-flow for Robotic Applications*, Australasian Conference on Robotics and Automation. 1-10, 2014.

3. Adarve, JD. and Mahony, R., *A Filter Formulation for Computing Real Time Optical Flow*, IEEE Robotics and Automation Letters, Vol. 1, 1192-1199, 2016.

4. Adarve, JD. and Mahony, R., *Spherepix: a Data Structure for Spherical Image Processing*, IEEE Robotics and Automation Letters, Vol. 2, 483-490, 2016.

Under review

1. Adarve, JD. and Mahony, R., *Real-time Structure Flow*, IEEE Transactions on Robotics.

1.3 Software Packages

1. **Optical-flow-filter:** https://github.com/jadarve/optical-flow-filter. Implementation of the optical flow filtering algorithm in CUDA. The repository provides demo applications to run the algorithm from images captured from a webcam (OpenCV based) and from a high-speed Basler USB3 camera.

2. **Spherepix:** https://github.com/jadarve/spherepix. Implementation of the Spherepix data structure for processing of spherical images.

3. **Structure-flow-filter:** (To be released). Implementation of the structure flow filtering algorithm in CUDA.

1.4 Thesis Outline

The thesis is divided into 5 chapters including this introduction. Chapters 2, 3 and 4 are the contribution chapters. These chapters are self-contained in terms of relevant literature, mathematical development and experimental results. Chapter 5 provides the general conclusions of the thesis as well as future work directions.

- **Chapter 2** presents the filtering approach for the computation of dense optical flow in real time. It provides a review of optical flow algorithms, in particular real-time methods used in robotic applications. The results section provides ground truth validation of the algorithm as well as qualitative comparison on high-speed 300 Hz real-life video taken in a driving scenario.
- **Chapter 3** describes the *Spherepix* data structure for efficient implementation of computer vision routines on spherical images. The literature review focuses on general-purpose computer data structures for the discretization of the sphere, as well as the state of the art on computer vision algorithms on spherical images. The mathematical concepts in this chapter are fundamental to understand the numerical implementation of the Structure Flow algorithm in Chapter 4.
- **Chapter 4** develops the *Structure Flow* field. The literature cited in this chapter focuses on the estimation of 3D motion fields from image or depth measurements (scene flow). This chapter develops the partial differential equations modeling the spatio-temporal evolution of the structure flow field on spherical camera geometry. Additionally, a filtering algorithm for the estimation of structure flow in real time is formulated and evaluated on ground-truth and real-life video sequences.
- **Chapter 5** provides the conclusions of the thesis and enumerates possible future research directions. The chapter also looks at the current challenges for the development of high-speed vision algorithms and their deployment on real-life systems.

Chapter 2

A Filter Formulation for Computing Real-time Optical Flow

Keep It Simple - Run It Fast

Juan Adarve

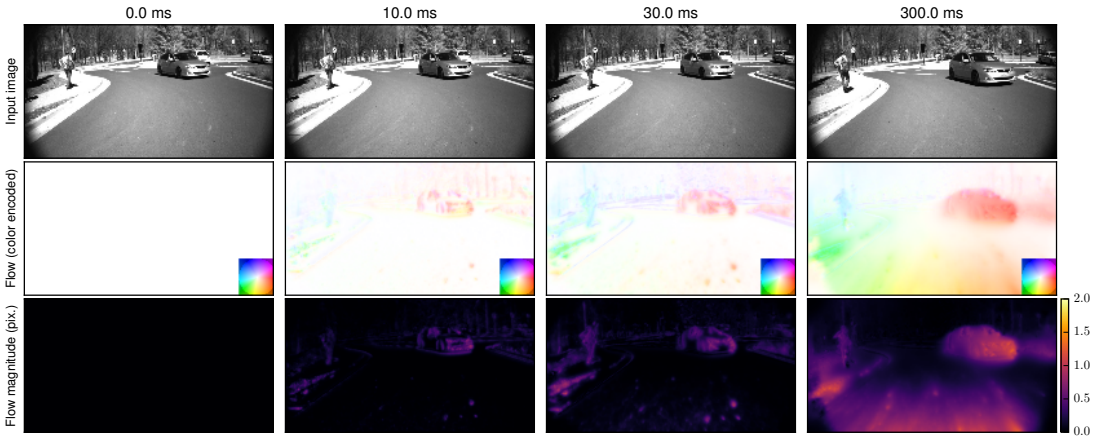


Figure 2.1: 300 Hz incremental computation of optical flow. Images captured with a high-speed camera are used to extract the underlying optical flow of the scene.

Optical flow is the two-dimensional vector field describing the motion of pixels in the image plane due to the motion of the environment and the camera. Optical flow offers important visual cues to a robotic vehicle moving in a dynamic environment. Applications such as visual servoing Hamel and Mahony [2002], vehicle landing Herisse et al. [2012], height regulation Ruffier and Franceschini [2005] and obstacle avoidance Srinivasan [2011a] use the estimated optical flow computed by the onboard vision system of the robot.

In a robotics context, an important requirement of any optical flow algorithm is its capability to run at real-time frequencies. In practice, this means that the vision processing system should run at the same frequency, or preferably faster, than the vehicle controller. For a typical high-performance autonomous vehicle, this translates to frame rates in the order of 200Hz Bangura et al. [2015]. Furthermore, typical robotic platforms have weight limitations that con-

strain the amount of compute hardware they can carry. For example, small aerial vehicles use similar embedded System on Chip (SoC) as those found in modern smart-phones. These chips are equipped with multi-core processing units as well as mobile Graphics Processing Units (GPU).

Real-time computation and embedded hardware constraints requires a different approach to compute optical flow from the state of the art algorithms documented in the well known benchmarks such as Baker et al. [2011]; Butler et al. [2012]; Geiger et al. [2013]. Typical algorithms from the Computer Vision community use two frames to estimate dense optical flow between the two images Baker et al. [2011]. In order to achieve highly accurate results, modern algorithms use sophisticated mathematical models to extract the optical flow from two frames. Typically, the most accurate algorithms are also the more computationally expensive, as it can be verified in the runtime reported in benchmark datasets cited above.

Recall the properties of a Robotic Embedded Vision system (REV) described in chapter 1, in particular the *high-speed video sampling* of the environment to capture the dynamics of objects in the world at high frequencies. This chapter proposes a new optical flow algorithm capable of running at frequencies as high as 800Hz at 640×480 on a Desktop computer and near 100 Hz on embedded GPU SoC at 320×240 pixel resolution.

The proposed algorithm follows a prediction-update filtering approach, where an internal optical flow state is incrementally built and updated using the stream of image data from the camera. An important component of the algorithms is the prediction stage, which is modeled as a system of partial differential equations to integrate forward in time the current estimation of image brightness and optical flow to create predictions for future time. Numerical solution to these equations is implemented using an efficient finite difference method based on upwind differences Thomas [1995]. This numerical method can be efficiently implemented on both GPU and FPGA hardware.

2.1 Outline

This chapter is divided as follows. Section 2.2 provides the relevant literature on optical flow and real-time algorithms. Section 2.3 develops the idea of incremental computation of optical flow. Section 2.4 explains the details of the filtering algorithm. Section 2.5 develops the numerical implementation of the propagation stage using finite difference methods. Section 2.6 provides the experimental results on both synthetic and real-life high-speed image sequences. Finally, the chapter is closed in Section 2.7 with some summary comments.

2.2 Background

Optical flow is the two-dimensional vector field describing the velocity of each pixel in a sequence of images. The computation of optical flow is one of the fundamental problems in computer vision and can be traced back beyond the seminal works of Lucas and Kanade [1981] and Horn and Schunck [1981].

Early research works on optical flow are summarized in the survey article by Barron [1994]. There, the authors classified optical flow methods according to the mathematical framework

used as: differential techniques, region based methods, energy based methods and phase based methods. From these, differential and region based methods are of interest in the context of this thesis.

- **Differential techniques:** Differential methods use spatio-temporal models of the image brightness to recover the underlying optical flow from a sequence of images. These methods are based on the brightness conservation assumption that states that the total amount of brightness in the image is conserved over time. Intuitively, this means that any change in the brightness value of a given pixel is due to the change in position of the object in the scene in that pixel direction.

  Early works in optical flow such as those of Lucas and Kanade [1981] and Horn and Schunck [1981] are differential methods. These two algorithms marked a distinction between local based (Lucas-Kanade) and global based methods (Horn-Schunck) for computing flow. Local based methods use a small support window around each pixel in the image to estimate optical flow, while global based methods impose a global constraint over the optical flow field to improve the quality of the estimation. In general, local based methods are faster than global based ones. A detailed explanation of these algorithms is provided in Section 2.2.2.

  Modern differential algorithms Brox et al. [2004]; Bruhn et al. [2005]; Werlberger [2012] utilize the same differential principles together with robust optimization frameworks to create highly accurate flow estimates. In particular, global based methods often use the L1 norm instead of the L2 (as in Horn and Schunck) to be more robust to outliers in the estimation process.

- **Region based algorithms:** Region based methods to compute optical flow can be thought as a search process to find a patch of texture in the second image that matches a reference patch in the first. A key difference of this approach compared to differential techniques is that it does not make the assumption of the image brightness to be differentiable.

  Early works such as that of Anandan [1989] uses the *sum of square differences* (SSD) to match texture patches within in a search region for each pixel. In order to support large pixel displacements, the algorithm is formulated in a pyramidal structure where coarse estimates of flow are computed on low resolution images and then are refined using higher resolution data. Srinivasan image interpolation algorithm follows a similar approach to find optical flow Srinivasan [1994], and has been used in real-life robotic systems Srinivasan [2011a].

  Modern algorithms such as SimpleFlow by *et. al.* Tao et al. [2012], PatchMatch by Bao et al. [2014], Piecewise Parametric Flow by Yang and Li [2015] and many others have proven the effectiveness of region based methods on public benchmarks.

More recent algorithms are listed on different benchmark datasets. The most relevant are: the Middlebury flow dataset by Baker et al. [2011][1], the Kitti dataset by Geiger et al. [2013][2] and the Sintel dataset by Butler et al. [2012][3]. Each of these datasets provide both test and

[1] http://vision.middlebury.edu/flow/
[2] http://www.cvlibs.net/datasets/kitti/eval_flow.php
[3] http://sintel.is.tue.mpg.de/

evaluation sequences to evaluate optical flow algorithms. The image sequences can be both real-life images, for which the ground truth is computed using fluorescent markers as in the Middlebury dataset or using odometry and laser measurements as in the Kitti dataset. The Sintel dataset is a purely synthetic dataset based on the Sintel open-source movie[4] created in Blender[5]. Ground truth optical flow can be extracted directly from the scene geometry (depth map) and the motion of objects relative to the camera.

Additionally, the survey article of Sun et al. [2014] offers a review of most recent algorithms together with a quantitative analysis of different optimization frameworks used to estimate optical flow. Also, the survey article of Chao et al. [2014] offers a list of optical flow algorithms currently used in robotic applications.

Optical flow algorithms are too numerous to create an exhaustive classification and review. Instead, the review provided in this chapter is focused on algorithms that have proven to work at real-time frequencies, thus making them effective for real-life robotic applications. In particular, differential optical flow algorithms are reviewed in detail, as such algorithm are closer to the mathematical formulation used in this chapter. Additionally, bioinspired algorithms as well as methods using alternative camera technologies such as *Dynamic Vision Systems* are included in this review.

2.2.1 Differential methods for optical flow computation

The most common algorithms to compute optical flow are the differential methods. These methods model the temporal change of intensity due to the underlying optical flow present in the image sequence. The approach is based on the well known *brightness conservation equation* that is the starting point of all differential methods Barron [1994].

Let $Y(\mathbf{p}, t) : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$ denote the image brightness at pixel position $\mathbf{p} \in \mathbb{R}^2$ and time $t \in \mathbb{R}$. The common assumption about image brightness is that the objects composing the scene are made of Lambertian materials. That is, the light scattered by the material is invariant to the viewer's view angle. Given this assumption, it is possible to say that the value of image brightness is constant over time. In other words, the total rate of change of image brightness over time is zero. Mathematically, one has

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = 0 \tag{2.1}$$

Since brightness is a function of independent variables $\mathbf{p}$ and t, one can decompose Equation (2.1) in terms of its partial derivatives as

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = \frac{\partial Y}{\partial \mathbf{p}} \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} + \frac{\partial Y}{\partial t} = 0 \tag{2.2}$$

where $\frac{\partial Y}{\partial \mathbf{p}} \in \mathbb{R}^2$ is the image gradient vector and $\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} \in \mathbb{R}^2$ is the relative change in pixel position, that is, the optical flow at $\mathbf{p}$. Equation (2.2) is the *brightness conservation equation* modeling the relationship between temporal brightness change and optical flow, and is the common starting point of all differential techniques.

[4] https://durian.blender.org/
[5] https://www.blender.org/

Equation (2.2) imposes only one constraint to the two-dimensional optical flow vector $\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t}$. That is, only the component of $\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t}$ parallel to the image gradient $\frac{\partial Y}{\partial \mathbf{p}}$ is relevant to (2.2). This is known as the *aperture problem* in the literature. Algorithms need to use some form of data integration or regularization to recover the full two-degrees of optical flow.

In differential methods, there are two families of algorithms to find the optical flow from a sequence of images. The first family, *local-based methods*, consists in algorithms that use a local support window around each image pixel to find the underlying optical flow. The second family, *global-based methods*, impose global constraints over the whole image that the resulting optical flow field must satisfy.

Local-based methods use a finite support window around each image pixel to find optical flow. Within this window, flow is assumed to be constant and all image data within the window is used to recover one single flow vector. The most popular algorithm of this family was proposed by Lucas and Kanade [1981]. Let $\Omega_{\mathbf{p}}$ denote a support window around pixel $\mathbf{p}$. The basic formulation of Lucas-Kanade method is to minimize cost function

$$\varepsilon_{\mathbf{p},\,\mathbf{\Phi}} = \sum_{\mathbf{q}\in\Omega_{\mathbf{p}}} \|\partial_{\mathbf{p}} Y_{\mathbf{q}}\mathbf{\Phi} + \partial_t Y_{\mathbf{q}}\|^2 \tag{2.3}$$

for the unknown optical flow $\mathbf{\Phi} := \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t}$ at $\mathbf{p}$. Here $\partial_{\mathbf{p}} Y_{\mathbf{q}}$ denotes the image gradient vector at pixel $\mathbf{q}$ and $\partial_t Y_{\mathbf{q}}$ is the image temporal derivative at $\mathbf{q}$.

A popular implementation of Lucas-Kanade method is the one developed by Bouguet [2001] and available in OpenCV[6]. To support large optical flow fields, this implementation uses a pyramidal approach to subdivide the problem in different image scales. At coarse scale, the algorithm computes a coarse estimation of flow from the low resolution data. This coarse estimate is refined using higher resolution data from the next scale level, until the original resolution level is reached.

Local based methods, in general, are subject to noise in the data within the support window. In regions of the image where there is no gradient information, that is, in textureless regions of the image, it is not possible to recover optical flow. Moreover, the estimated optical flow suffers from the aperture problem, where only the flow in the direction of the image gradient can be computed.

Global-based methods impose a prior constraint over the optical flow at each pixel. Typically, this prior has the form of a smoothness constraint where the optical flow field is assumed to be smooth across the image. One example of global based methods is the seminal work by Horn and Schunck [1981]. In their work, the authors formulate the optical flow problem as a variational problem where the brightness conservation equation is complemented with a flow smoothness term to regularize the computation of flow in image regions with poor texture data. The goal is to minimize cost functional in Equation (2.4) with respect to $\mathbf{\Phi}$.

$$\varepsilon^2 = \iint \|\partial_{\mathbf{p}} Y\mathbf{\Phi} + \partial_t Y\|^2 + \alpha^2 \left[(\partial_x\phi_x)^2 + (\partial_y\phi_x)^2 + (\partial_x\phi_y)^2 + (\partial_y\phi_y)^2\right] dxdy \tag{2.4}$$

Equation (2.4) can be solved using calculus of variations, expressing the integral in terms of its underlying Euler-Lagrange system of partial differential equations. The resulting system

[6] http://opencv.org/

couples the solution of each pixel and its neighbors, and requires an iterative method to find the solution (for example, Gauss-Jordan method).

The key advantage of global-based methods over local-based ones is that estimations at each pixel are well defined thanks to the smoothness term. In regions of the image where image gradient information is poor, the smoothness term in Equation (2.4) dominates over the image term. Consequently, the flow in those regions will be filled with flow coming from regions with high texture content such as image edges.

The work of Horn and Schunck opened a new research field in computer vision for the computation of optical flow using variational methods. One of such algorithms is proposed by Brox *et. al.* Brox et al. [2004] which is used as reference in the experimental evaluation in Section 2.6. Global-based methods typically outperform local-based methods in terms of accuracy in benchmarks such as Middlebury, Kitti and Sintel. However, local based methods perform better in terms of runtime, as it will be illustrated in the next section.

2.2.2 Real-time algorithms and systems

This section describes real-time optical flow algorithms and systems. This separation from mainstream optical flow algorithms is important in a robotics context, where algorithm's runtime is as important as its accuracy.

In general, real-time flow algorithms are local-based methods. Many of these methods are some form of Lucas-Kanade method Lucas and Kanade [1981], region based method, like Srinivasan [1994], or neuromorphic approaches Mueggler et al. [2014]. Image data is constrained to a local support window around each pixel. Moreover, the algorithms are designed such that the flow estimation is independent for each pixel, thus allowing one to perform computations in parallel for each all pixels. This pixel independence allows algorithm implementation on multi-core processors or Graphic Processing Units (GPU). Moreover, it is possible to create digital circuit pipelines to compute optical flow directly in silicon. Prototypes of these pipelines can be deployed and tested on Field Programmable Gate Arrays (FPGA) chips.

Table 2.1 lists relevant real-time dense optical flow methods found in the literature. For each algorithm, its resolution (pixels), working framerate (Hz) and throughput (Mpix/s) is reported. The throughput is computed as

$$\text{Tput} = \frac{\text{resolution} \cdot \text{framerate}}{10^6} \tag{2.5}$$

Typically, real-time optical flow algorithms are implemented on GPU hardware to take advantage of the parallel compute power to perform per-pixel operations simultaneously. Bouguet [2001] created a pyramidal implementation of the Lucas-Kanade optical flow method freely available in OpenCV both in CPU and GPU versions. The GPU version is used in Section 2.6 to compare both the accuracy and performance of the optical flow filter algorithm. The eFOLKI algorithm of Plyer et al. [2014] is an implementation of the Lucas-Kanade method that increases the robustness of the algorithm by applying a Rank-n transform Zabih and Woodfill [1994] before minimizing a SSD cost function to find the optical flow.

FPGA systems to compute optical flow rely in deep compute pipelines to perform calculations as pixel data enters the FPGA chip either directly from an image sensor or from an

external memory bank. The systems by Plett et al. [2012] and Zhang et al. [2008] are inspired by the visual cortex of insects to compute local motion descriptors.

	Resolution	**Framerate (Hz)**	**Tput. (Mpix/s)**	**Hardware**
Adarve and Mahony [2016a]	640×480	814	250.06	GPU
Derome et al. [2016]	1242×375	10	4.65	GPU
Kroeger et al. [2016]	128×54	600	4.14	CPU
Plyer et al. [2014]	640×480	166	60	GPU
Barranco et al. [2013]	640×480	31	9.52	FPGA
Plett et al. [2012]	240×240	350	20.16	FPGA
Anguita et al. [2009]	1280×1016	68.8	89.47	CPU
Zhang et al. [2008]	256×256	320	20.97	FPGA
Farnebäck [2003]	640×480	27	8.29	GPU
Bouguet [2001]	640×480	33	10.13	GPU

Table 2.1: Real-time optical flow methods. Image resolutions and framerate were copied from the original papers.

2.2.3 Neuromorphic approaches

There are alternative image sensors and algorithms inspired by nature from which it is possible to extract optical flow. These technologies and algorithms are of particular importance to robotics as they offer fast visual feedback to robotic vehicles.

One of such approaches comes from understanding the visual system of insects. Insects such as flies and bees have compound eyes made of a grid of light sensors known as the ommatidia. Each ommatidium captures a narrow field of view of the scene and is connected to the visual cortex of the insect. The reader is encouraged to read the review paper by Borst on the structure of insect's eye Borst [2009]. It is possible to model the correlation between neighboring ommatidium to extract motion information. This correlation model is known as the Reichardt motion detector Reichardt [1987]. Systems such as those by Zhang et al. [2008] and Plett et al. [2012] have demonstrated the effectiveness of this motion detector in real-life systems.

The work by Srinivasan [2011a,b] offers an insight on how the motor control part of the insects' neural cortex connects to the visual system. In his experiments, he has demonstrated how honeybees use optical flow for tasks such as navigation, obstacle avoidance and landing. The understanding of insect vision has brought insight to robotics to solve the same tasks on flying vehicles. Such is the case of the works by Ruffier and Franceschini on optical flow regulation Ruffier and Franceschini [2005], the work of Mafrica *et. al.* on velocity and steering angle control of a car vehicle Mafrica et al. [2016] and the work by Herisse et al. [2012] on landing and take-off of an aerial vehicle from a moving platform using optical flow.

Another technology that has been proven effective for robotic vision are the *Event-based Cameras*. These cameras are a complete paradigm shift compared to standard digital camera hardware, with the principal difference lying in the information transmitted to the user. As their name suggests, event cameras transmit events occurring in the scene instead of full

image frames. An event is triggered by a temporal change in pixel brightness. Current hardware[7] transmits event location, timestamp and polarity (plus or minus) asynchronously. Consequently, the bandwidth required between camera and processing units is significantly reduced, as the camera only transmits the difference between two images. Applications of this technology are emerging: Benosman et al. [2012, 2014] extract optical flow from the event stream using a differential framework similar to that used on standard gray-scale images. Mueggler et al. [2014] use the stream of events to track the 6-DOF pose of an aerial vehicle using the on-board vehicle's CPU.

2.3 Incremental Computation of Optical Flow

The main concept in the development of the proposed real-time optical flow algorithm is that of incrementally building a dense flow estimate over time. In this approach, one has an internal optical flow state that is constantly refined using new image measurements from the camera. Instead of computing dense flow fields from two images, as typical computer vision algorithms do, an incremental approach will exploit the large availability of data one has in a real-life robotic vision system to constantly estimate optical flow.

This incremental approach has two advantages over standard algorithms. First, the optical flow field is temporally smooth. Thanks to the incremental nature of the algorithm, the temporal evolution of the estimated flow field will show a smooth transition between two consecutive image frames. Second, there are computational efficiencies in this approach. Instead of using a complex algorithm to compute dense optical flow fields from two images, one can design a simpler flow update algorithm considering new image data and the current flow state estimation.

Intrinsic to the incremental approach to compute optical flow, is the concept of temporal evolution of the flow field. At each time step, that is, when a new image arrives, one needs to propagate forward in time the old flow state to align it to current time. Once the prediction is completed, both the flow state and the measurements are temporally aligned, and a new estimate of optical flow can be created using both pieces of information.

This process matches a filter architecture consisting of update and prediction stages. To the best of the author's knowledge, this architecture for computing optical flow was first described in the PhD dissertation of Black [1992]. In his work, Black uses robust statistic methods in the update stage to refine the predicted optical flow from previous time step. The prediction is formulated as the warping of the optical flow field forward in time. This warping can be implemented by adding the optical flow field to each pixel coordinate and then performing a re-sampling of the resulting image.

The proposed algorithm in this chapter follows the same general update-prediction architecture. However, a key novelty is the formulation of the prediction stage as system of partial differential equations modeling the transport of optical flow, and any associated field, by the optical flow. The prediction stage is implemented by numerically solving these transport PDEs using a finite difference solver. In contrast to image warping, the finite difference solver does work on a fixed grid of points and hence, it does not require a re-sampling post-processing

[7] http://inilabs.com/products/dynamic-vision-sensors/

stage.

As it will be shown throughout the chapter, this filter formulation leads to regular computations that can be realized in both GPU and FPGA hardware, and can run at the frequencies required by a robotic vision system.

2.4 Filter Architecture

The optical flow filter algorithm is constructed as a pyramid of filter loops. Let H be the number of levels of the pyramidal structure, and $h = 1, \dots H$ the level index. The filter state at time index k is the pyramid of vector fields $\mathbf{X}^k = \left\{ {}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k}}, {}^{\mathrm{H\text{-}1}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k}}, \dots, {}^{1}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k}} \right\}$. The state at level H is the optical flow field ${}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k}}(\mathbf{p}) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ estimated using low resolution image data. For levels $h = H - 1, \dots, 1$ the state is defined as the vector field ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k}}(\mathbf{p}) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ that refines the optical flow from the level immediately above using higher resolution data available at level h.

The reconstruction of optical flow at lower levels is achieved by iterating Equation (2.6) for $h = H - 1, \dots, 1$

$$ {}^{\mathrm{h}}\mathbf{\Phi}^{\mathrm{k}} = 2\, {}^{\mathrm{h+1:h}}\mathbf{\Phi}^{\mathrm{k}} + {}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k}} \tag{2.6} $$

where ${}^{\mathrm{h+1:h}}\mathbf{\Phi}^{\mathrm{k}}$ denotes the upsampling of flow from level $h+1$ to h. This upsampling is implemented by linear interpolation of ${}^{\mathrm{h+1}}\mathbf{\Phi}^{\mathrm{k}}$. The interpolated flow is then multiplied by the original down-sampling factor 2 to scale the flow to the new image resolution.

The filter algorithm operates cascading information from top level H down to level 1. At time k, the propagation stage takes place for all levels of the pyramid. In this stage, a prediction of the state variable for time $k+1$ is calculated based on current estimation. The propagated state is denoted as $\mathbf{X}^{k+} = \left\{ {}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+}}, {}^{\mathrm{H\text{-}1}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+}}, \dots, {}^{1}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+}} \right\}$, where superscript $k+$ refers to the predicted state for time $k+1$ before the update stage takes place. Once the propagation is completed, the predicted state is combined with the newly computed brightness parameters at each level to create an updated state $\mathbf{X}^{k+1} = \left\{ {}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+1}}, {}^{\mathrm{H\text{-}1}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+1}}, \dots, {}^{1}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+1}} \right\}$.

The filter pyramid consists of two types of filter loops. Figure 2.2 illustrates the filter loop at top level H. Brightness model parameters $\{ {}^{\mathrm{H}}\partial_{\mathbf{p}}\hat{Y}^{\mathrm{k+1}}, {}^{\mathrm{H}}\hat{Y}^{\mathrm{k+1}} \}$ are extracted from input image ${}^{\mathrm{H}}Y^{\mathrm{k+1}}$, and form the measurement data for the filter loop, formed by update and propagation stages. In the propagation stage, the old estimate of coarse optical flow ${}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k}}$ is propagated to create a prediction ${}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+}}$ for time $k+1$. This prediction, together with the new brightness parameters are used in the update block to create new flow estimate ${}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+1}}$.

The second type of filter loop is illustrated in Figure 2.3, and is used for all lower levels of the pyramid $h = H - 1, \dots, 1$. Brightness parameters $\{ {}^{\mathrm{h}}\partial_{\mathbf{p}}\hat{Y}^{\mathrm{k+1}}, {}^{\mathrm{h}}\hat{Y}^{\mathrm{k+1}} \}$ are estimated from input image ${}^{\mathrm{h}}Y^{\mathrm{k+1}}$. Equation (2.6) is used to reconstruct the flow at level h given the flow ${}^{\mathrm{h+1}}\mathbf{\Phi}^{\mathrm{k}}$ from one level above and current state ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k}}$. In the propagation, the old optical flow ${}^{\mathrm{h}}\mathbf{\Phi}^{\mathrm{k}}$ is used to propagate ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k}}$ and ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k}}$ to produce predictions ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+}}$ and ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k+}}$. Notice that ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k+}}$ represents a prediction of the image at $k+1$ given the old optical flow. Under the brightness constancy assumption, this prediction is expected to be close to the newly computed ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k+1}}$ parameter from raw image data. Thus, any difference between ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k+}}$ and ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k+1}}$ will be due to some small change, i.e., , ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+1}}$, between the new and old optical flows at this level. The purpose of the update stage is to find the increment ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+1}}$ that corrects the optical flow at this

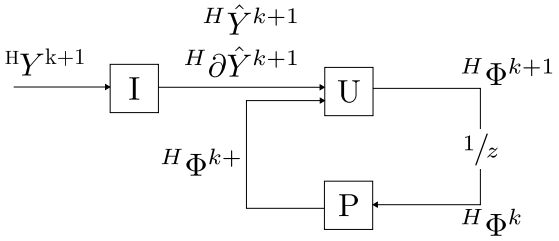


Figure 2.2: Top level filter loop. In the image preprocessing block [I], brightness parameters are estimated from the new image and enter the filter loop. In the propagation block [P], a prediction of optical flow for the next time iteration is created. In the update [U], this prediction together with the brightness parameters create a new estimation of the flow field.

level.

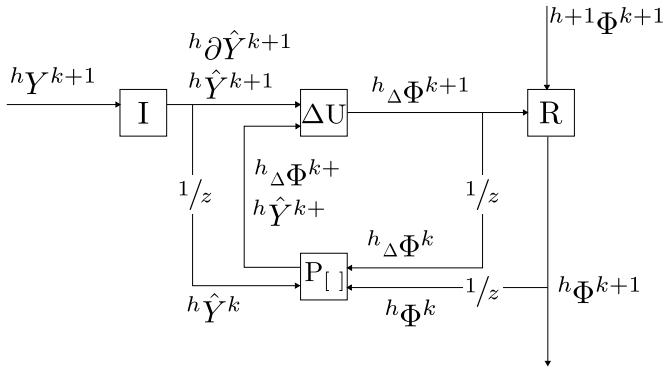


Figure 2.3: Lower levels filter scheme. In the reconstruction block [R], optical flow for this level is reconstructed using flow from level above. Brightness parameters from the new image are computed at [I]. In the Propagation block [$P_{[]}$], old flow field is used to propagate the old state and the constant brightness parameter. These predictions are combined with new brightness parameters at the Update block [ΔU] to create a new state estimate.

2.4.1 Extraction of brightness parameters

The first stage in the algorithm is to build an image pyramid $\{^{l}Y^{k}, \ldots, ^{H}Y^{k}\}$ provided the input image from the camera. The pyramid is constructed by successive low pass filtering the image by a Gaussian function and sub-sampling every second pixel.

At each pyramid level, and for each pixel $\mathbf{p}$, the parameters of a linear brightness model are extracted from the raw image $^{h}Y^{k+1}$. The linear brightness model is made of parameters $^{h}\hat{Y}_{p}^{k+1} \in \mathbb{R}^1$ for the constant term and $^{h}\partial_{\mathbf{p}}\hat{Y}_{p}^{k+1} \in \mathbb{R}^2$ for the brightness slope or gradient vector. Brightness parameters are determined as the arg min of cost function

$$\varepsilon_{p}^{h} = \sum_{\mathbf{q}\in\Omega_{\mathbf{p}}} w(\mathbf{p},\mathbf{q}) \left\| {}^{h}Y_{\mathbf{q}}^{k+1} - {}^{h}\partial_{\mathbf{p}}\hat{Y}_{p}^{k+1}(\mathbf{p}-\mathbf{q}) - {}^{h}\hat{Y}_{p}^{k+1} \right\|^2 \tag{2.7}$$

where $\Omega_{\mathbf{p}}$ denotes a support window around pixel $\mathbf{p}$. The weight function $w(\mathbf{p},\mathbf{q})$ is a Gaus-

sian function for the weight of data at pixel $\mathbf{q}$ relative to $\mathbf{p}$. In practice, a support window of size 5×5 centered on $\mathbf{p}$ is used and together with a Gaussian weight function computed as the 2D convolution of 1D weight mask $w = [1, 4, 6, 4, 1]/16$.

The cost function (2.7) describes a linear least square process in terms of unknown parameters ${}^{\mathrm{h}}\hat{Y}_p^{\mathrm{k+1}}$ and ${}^{\mathrm{h}}\partial_{\mathbf{p}}\hat{Y}_p^{\mathrm{k+1}}$. Given the symmetry of the cost function around $\mathbf{p}$, the extraction of the brightness parameters can efficiently be implemented as a series of 1D convolutions in the row and column axes. Figure 2.4 illustrates the sequence of convolutions to compute brightness parameters.

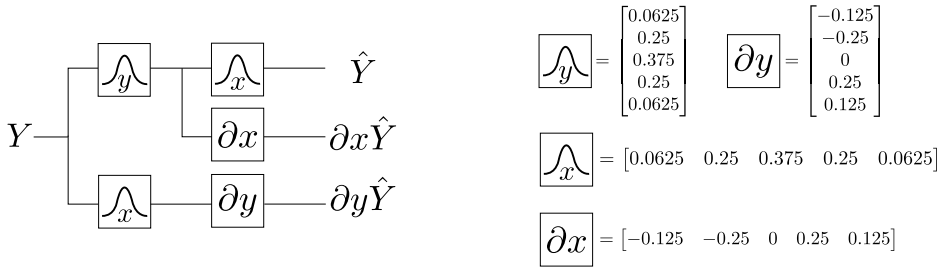


Figure 2.4: Brightness model parameters computation.

2.4.2 State propagation

Consider the optical flow field $\mathbf{\Phi}(\mathbf{p}, t) : \mathbb{R}^2 \to \mathbb{R}^2$ in continuous pixel coordinate $\mathbf{p} \in \mathbb{R}^2$ at time t. Let $\mathbf{p}(0)$ be the initial pixel position and $\mathbf{\Phi}(\mathbf{p}(0), 0)$ be the optical flow at $\mathbf{p}$ at time zero. For the derivation of the propagation equations modeling the temporal evolution of brightness and optical flow, it is assumed that the optical flow field $\mathbf{\Phi}(\mathbf{p}, t)$ is constant over time. That is[8]

$$\frac{\mathrm{d}\mathbf{\Phi}(\mathbf{p}(t), t)}{\mathrm{d}t} \approx 0 \tag{2.8}$$

Under this assumption, the evolution of pixel position $\mathbf{p}(t)$ is given by the ordinary differential equation

$$\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = \mathbf{\Phi}(\mathbf{p}(t), t) \tag{2.9}$$

That is, the pixel velocity at time t is equal to the optical flow at that pixel location. Given the initial conditions $\mathbf{p}(0)$ and the optical flow $\mathbf{\Phi}(\mathbf{p}(0), 0)$, one can directly compute the pixel position at future time by solving ODE (2.9).

For the derivation of the optical flow transport equation, it is convenient to think about the optical flow field as a vector attached to pixel $\mathbf{p}(t)$. That is, to assume that the optical flow moves along with pixel $\mathbf{p}(t)$. At any point in time, the optical flow associated to $\mathbf{p}(t)$ is the

[8] While the transport equations are correct under the assumption of constant flow and pixel motion in the image plane, they do not explicitly model the underlying kinematics of points in the scene projected onto the image plane. The correct formulation of the propagation of optic flow requires the concept of structure flow (Chapter 4), and is one of the principal contributions of the present thesis. The present assumption, however, provides a reasonable approximation and works well in practice, where the error in the propagation is dominated by the correction provided by the innovation.

same and is fully characterized by the initial conditions of the problem.

$$\mathbf{\Phi}(\mathbf{p}(t), t) \approx \mathbf{\Phi}(\mathbf{p}(0), 0) \tag{2.10}$$

Computing $\frac{\mathrm{d}}{\mathrm{d}t}$ of (2.10) one obtains the constant flow assumption made in Equation (2.8) and these two equations are equivalent. Since optical flow is a vector field depending on pixel position and time, its evolution model is given by decomposing Equation (2.8) in terms of its partial derivatives. Mathematically, one has

$$\frac{\mathrm{d}\mathbf{\Phi}(\mathbf{p}(t), t)}{\mathrm{d}t} = \frac{\partial \mathbf{\Phi}}{\partial \mathbf{p}} \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} + \frac{\partial \mathbf{\Phi}}{\partial t} \approx 0 \tag{2.11}$$

where $\frac{\partial \mathbf{\Phi}}{\partial \mathbf{p}} \in \mathbb{R}^{2\times 2}$ is the Jacobian matrix of $\mathbf{\Phi}$ at position $\mathbf{p}$. Replacing the pixel velocity $\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t}$ by the optical flow value at $\mathbf{p}$, one obtains *Partial Differential Equation* (2.12) modeling the transport of the optical flow field by the optical flow field itself[9].

$$\frac{\mathrm{d}\mathbf{\Phi}(\mathbf{p}(t), t)}{\mathrm{d}t} = \frac{\partial \mathbf{\Phi}}{\partial \mathbf{p}} \mathbf{\Phi} + \frac{\partial \mathbf{\Phi}}{\partial t} \approx 0 \tag{2.12}$$

Equation (2.12) belongs to the family of non-linear hyperbolic partial differential equations used to model several types of transport processes. See the book of LeVeque [2002] for several examples of such transport phenomena.

Following a similar approach, it is possible to model the transport of an arbitrary scalar field $c(\mathbf{p}, t) : \mathbb{R}^2 \to \mathbb{R}$ by the optical flow. The basic assumption is the conservation of $c(\mathbf{p}, t)$ in time. That is

$$\frac{\mathrm{d}c}{\mathrm{d}t} = 0 \tag{2.13}$$

Since c is a function of both position and time, one can decompose Equation (2.13) in terms of its partial derivatives as

$$\frac{\mathrm{d}c}{\mathrm{d}t} = \frac{\partial c}{\partial \mathbf{p}} \mathbf{\Phi} + \frac{\partial c}{\partial t} = 0 \tag{2.14}$$

where $\frac{\partial c}{\partial \mathbf{p}} \in \mathbb{R}^2$ is the gradient vector of c at $\mathbf{p}$. Equation (2.14) has the same formulation as the *brightness conservation equation* (2.2) derived in Section 2.2 and can be used to create predictions of a field c such as image brightness.

Propagation equations for the filter algorithm

Equations (2.12) and (2.14) are the fundamental PDEs that model the propagation of the filter state variables in the algorithm.

For top level $h = H$, the propagation of the low resolution optical flow state ${}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k}}$ is

[9] An example in which Equation (2.12) approximately holds to zero is when the camera moves along the focal axis direction, thus inducing a divergent optical flow field such as that in Figure 1.1b. In such case, the optical flow vectors magnitude grows as it travels from the focus of expansion to the image sides. Thus, PDE (2.12) would require an extra term to inject (or remove) energy from the optical flow based on the camera and the environment kinematics. However, in practice this energy is small and can be injected to the estimated flow field at the update stage of the algorithm

governed by an instance of Equation (2.12) acting on it. That is

$$\frac{\partial {}^{\mathrm{H}}\mathbf{\Phi}}{\partial \mathbf{p}}\,{}^{\mathrm{H}}\mathbf{\Phi} + \frac{\partial {}^{\mathrm{H}}\mathbf{\Phi}}{\partial t} = 0 \tag{2.15}$$

Provided with initial conditions ${}^{\mathrm{H}}\mathbf{\Phi}(0) := {}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k}}$ at time $t = 0$, one is interested in finding a solution ${}^{\mathrm{H}}\mathbf{\Phi}(1) =: {}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+}}$ at time $t = 1$ corresponding to the propagated flow field for next time step.

For lower levels $h = 1, \ldots, H-1$, the propagation stage models the transport of state ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k}}$ and brightness constant parameter ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k}}$ by the optical flow field ${}^{\mathrm{h}}\mathbf{\Phi}^{\mathrm{k}}$ computed in Equation (2.6), which in turn is being transported by itself. This is modeled by the system of PDEs

$$\frac{\partial {}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}}{\partial \mathbf{p}}\,{}^{\mathrm{h}}\mathbf{\Phi} + \frac{\partial {}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}}{\partial t} = 0 \tag{2.16}$$

$$\frac{\partial {}^{\mathrm{h}}\hat{Y}}{\partial \mathbf{p}}\,{}^{\mathrm{h}}\mathbf{\Phi} + \frac{\partial {}^{\mathrm{h}}\hat{Y}}{\partial t} = 0 \tag{2.17}$$

$$\frac{\partial {}^{\mathrm{h}}\mathbf{\Phi}}{\partial \mathbf{p}}\,{}^{\mathrm{h}}\mathbf{\Phi} + \frac{\partial {}^{\mathrm{h}}\mathbf{\Phi}}{\partial t} = 0 \tag{2.18}$$

The initial conditions of the system at time $t = 0$ are set to ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}(0) := {}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k}}$, ${}^{\mathrm{h}}\hat{Y}(0) := {}^{\mathrm{h}}\hat{Y}^{\mathrm{k}}$ and ${}^{\mathrm{h}}\mathbf{\Phi}(0) := {}^{\mathrm{h}}\mathbf{\Phi}^{\mathrm{k}}$. The PDE system is solved for time $t = 1$ for ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}(1) =: {}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+}}$ and ${}^{\mathrm{h}}\hat{Y}(1) =: {}^{\mathrm{h}}\hat{Y}^{\mathrm{k+}}$. Notice that, while there is a solution ${}^{\mathrm{h}}\mathbf{\Phi}(1)$ for the optical flow, this solution never abandons the propagation block of the filter (Figure 2.3), and it can be regarded as an internal state in the propagation.

The numerical details for the solution of these equations are provided in Section 2.5

2.4.3 State update

In the update stage, a new state estimate $\mathbf{X}^{k+1}$ is produced by correcting the predicted state $\mathbf{X}^{k+}$ coming from the propagation stage using new image data captured by the camera.

The update stage is formulated as an independent least squares minimization problem for each pixel in the image. The least squares cost function for each pixel is composed of two terms: a data term that uses brightness parameters to extract a new optical flow estimate and a temporal smoothness term that uses the predicted flow state as a prior solution to the problem. The formulation of the cost function varies between the top level in the pyramid and the levels below.

For level H, the predicted optical flow ${}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+}}$ is updated using brightness parameters ${}^{\mathrm{H}}\hat{Y}^{\mathrm{k+1}}$ and ${}^{\mathrm{H}}\partial_{\mathbf{p}}\hat{Y}^{\mathrm{k+1}}$. The update cost function at pixel $\mathbf{p}$

$$\varepsilon^{\mathrm{H}}_{\mathbf{p}} = \left\| {}^{\mathrm{H}}\partial_{\mathbf{p}}\hat{Y}^{\mathrm{k+1}}_{\mathbf{p}} \cdot {}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+1}}_{\mathbf{p}} + {}^{\mathrm{H}}\hat{Y}^{\mathrm{k+1}}_{\mathbf{p}} - {}^{\mathrm{H}}\hat{Y}^{\mathrm{k}}_{\mathbf{p}} \right\|^2 + {}^{\mathrm{H}}\gamma \left\| {}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+1}}_{\mathbf{p}} - {}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+}}_{\mathbf{p}} \right\|^2 \tag{2.19}$$

consists of two terms. First, a data term relates the old image, characterized by ${}^{\mathrm{H}}\hat{Y}^{\mathrm{k}}_{\mathbf{p}}$, and the new image with brightness parameters ${}^{\mathrm{H}}\hat{Y}^{\mathrm{k+1}}_{\mathbf{p}}$ and ${}^{\mathrm{H}}\partial_{\mathbf{p}}\hat{Y}^{\mathrm{k+1}}_{\mathbf{p}}$. This term is a one-pixel version of the Lucas-Kanade cost function (2.3). The second term is a regularization term that includes the predicted state ${}^{\mathrm{H}}\mathbf{\Phi}^{\mathrm{k+}}_{\mathbf{p}}$ as a prior solution to the least-squares problem. A scalar parameter ${}^{\mathrm{H}}\gamma$

controls the weight of this term relative to the data term. In textureless regions of the image, where ${}^{\mathrm{H}}\partial_{\mathbf{p}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+1}}$ is nearly zero, and thus it is not possible to estimate ${}^{\mathrm{H}}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+1}}$, the predicted flow ${}^{\mathrm{H}}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+}}$ will act as best solution for time $k+1$.

After some algebraic manipulation, Equation (2.19) can be expressed as a linear system of the form

$$M\,{}^{\mathrm{H}}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+1}} = q \tag{2.20}$$

where

$$M = ({}^{\mathrm{H}}\partial_{\mathbf{p}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+1}})({}^{\mathrm{H}}\partial_{\mathbf{p}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+1}})^{\top} + {}^{\mathrm{H}}\gamma I_{2\times 2} \tag{2.21}$$

and

$$q = {}^{\mathrm{H}}\gamma\,{}^{\mathrm{H}}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+}} + {}^{\mathrm{H}}\partial_{\mathbf{p}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+1}}({}^{\mathrm{H}}\hat{Y}_{\mathbf{p}}^{\mathrm{k}} - {}^{\mathrm{H}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+1}}) \tag{2.22}$$

Given the small size of the linear system, it is advantageous to use the adjoint representation of the matrix inverse. Thus, solving the linear system yields

$${}^{\mathrm{H}}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+1}} = \frac{1}{\det(M)}\mathrm{adj}(M)q \tag{2.23}$$

Notice that if ${}^{\mathrm{H}}\gamma > 0$, then

$$\det(M) := {}^{\mathrm{H}}\gamma\left({}^{\mathrm{H}}\gamma + \|{}^{\mathrm{H}}\partial_{\mathbf{p}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+1}}\|\right) \neq 0 \tag{2.24}$$

and hence, solution to Equation (2.23) is well conditioned for all pixels in the image. Textureless regions of the image will simply use the predicted flow ${}^{\mathrm{H}}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+}}$ as best solution for ${}^{\mathrm{H}}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+1}}$ at time $k+1$.

For lower levels of the pyramid $h = H-1,\ldots,1$, the update stage aims to correct state prediction ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+}}$ using new image brightness information. Recall from Section 2.4 that ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+1}}$ represents an increment to the coarse optical flow ${}^{\mathrm{h+1}}\mathbf{\Phi}^{\mathrm{k+1}}$, given higher resolution image data at level h. Also, recall that ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k}}$ was propagated using ${}^{\mathrm{h}}\mathbf{\Phi}^{\mathrm{k}}$ (Equation (2.17)) and a prediction ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k+}}$ exists for time $k+1$. This prediction is expected to be similar to ${}^{\mathrm{h}}\hat{Y}^{\mathrm{k+1}}$ computed from image data, and, under the brightness conservation assumption, any difference between both parameters is due to some optical flow, in this case, state ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}^{\mathrm{k+1}}$.

The update cost function for ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+1}}$ at pixel $\mathbf{p}$ is defined as

$$\varepsilon_{\mathbf{p}}^{\mathrm{h}} = \left\|{}^{\mathrm{h}}\partial_{\mathbf{p}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+1}} \cdot {}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+1}} + {}^{\mathrm{h}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+1}} - {}^{\mathrm{h}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+}}\right\|^2 + {}^{\mathrm{h}}\gamma\left\|{}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+1}} - {}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+}}\right\|^2 \tag{2.25}$$

Similar to the update at top level, Equation (2.25) consists of two terms: a data term to relate new brightness parameters with the predicted ${}^{\mathrm{h}}\hat{Y}_{\mathbf{p}}^{\mathrm{k+}}$ to generate a new ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+1}}$ estimate and a temporal regularization term to introduce ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}_{\mathbf{p}}^{\mathrm{k+}}$ as a prior solution for regions in the image with poor gradient information. Solution to Equation (2.25) follows the same linear system formulation and solution as Equation (2.19) at level H.

State smoothing

To spread the new estimate of optical flow to regions of the image with poor texture information, a smoothing filter is applied to the output of the update block to diffuse the estimates of optical flow computed from regions with high image gradient information. In the current version of the algorithm, this diffusion mechanism is implemented as a simple 5×5 average window applied to the updated optical flow. The number of iterations of this average filter is a user-defined parameter, where 4 iterations is a reasonable value.

2.5 Numerical Implementation of State Propagation

This section presents all the details for the numerical implementation of the state propagation stage described in Section 2.4.2.

The study of numerical methods for solving partial differential equations is a complete branch of applied mathematics in itself (see LeVeque [2002] and Thomas [1995, 1999] for an introduction to numeric PDE theory). Numerous numerical methods exists for different types of applications such as computational fluid dynamics, aerodynamics and weather modeling. In general, the choice of a particular numerical method is problem and application dependent.

Numerical methods can be classified according to their approach to derive a discrete version of the underlying continuous PDE problem. Methods can be divided as *finite volume methods* and *finite difference methods*.

Finite volume methods formulate the partial differential equation problem in terms of a *conservation law*. First, the discrete volume, or area, element is defined. Simple examples of these elements are a cube and a square. This elements act as reservoirs containing a certain amount of "material" at any given time. Then, for the derivation of the conservation law, one needs to define the flux of material entering and leaving the volume elements on each of its sides or faces. These flux equations form the conservation law specific to the problem and can be discretized to create a numeric implementation. The reader is encouraged to look at the book of LeVeque [1992] for an introduction to finite volume methods.

Finite volume methods tend to be more physically accurate than finite difference methods. One reason for this is the modeling process in terms flux processes, which naturally matches the physical properties of the real-life phenomena. This gain in accuracy, as is usually the case, comes at the price of computational complexity in order to actively balance the input and output material flux at each discrete time step.

Finite difference methods offer a more direct approach to solve a partial differential equation. Basically, one only needs to define the volume or area element (same as finite volume methods), and define discrete operators for each partial derivative present in the PDE. Depending on the type of equation (parabolic, elliptic or hyperbolic), one needs to carefully choose the difference operators to avoid instability in the numerical scheme. Hyperbolic PDEs, which model transport process (the optical flow state propagation), are very sensible to this choice of operators. The books by Thomas Thomas [1995, 1999] are an excellent reference to study the different types of PDE and their numerical solution.

The following are fundamental properties of any numerical scheme used for propagating optical flow:

- **Robustness to input noise:** the optical flow extracted from images is a noisy vector field. Numerical schemes should be able to handle this noise.
- **Parallelizable:** the numerical methods should be parallelizable at a pixel granularity. Operations for each pixel should require a small support window of data around it to perform calculations. These properties are fundamental for an efficient implementation on GPU or FPGA hardware.

Other properties such as long-term accuracy of the propagated output are less important. Recall from the filter architecture section that the propagated optical flow is refreshed with new image data coming from the camera. Consequently, the initial conditions of the numerical scheme are reset at each time step and the propagation should be accurate just between two images. Given its simplicity and possibility of parallel implementations, the finite difference framework is chosen as the numerical scheme to implement the state propagation stage of the filtering algorithm.

2.5.1 Discrete state propagation equations

Let

$$\mathbf{\Phi}_{ij}^{\mathrm{n}} := \begin{pmatrix} u_{ij}^{\mathrm{n}} \\ v_{ij}^{\mathrm{n}} \end{pmatrix} \tag{2.26}$$

be the optical flow vector at discrete pixel location (i, j). In this section, index n refers to an internal time index in the propagation block, and it should not be confused with time index k used in the filter algorithm. Pixels (i, j) are equally spaced with pixel size

$$\Delta x = 1 \text{ pix} \tag{2.27}$$

and is the same for all pyramid levels.

To guarantee stability, the numerical method needs to run for several time iterations to reach the solution at final time. These iterations occur between two image frames, and should not be confused with the camera frame rate. Parameter N denotes the number of iterations of the scheme and $n = 0, \ldots, N-1$ is the iteration index. Each iteration covers an equal time step

$$\Delta t = \frac{1}{N} \tag{2.28}$$

The discrete version of the optical flow propagation in Equations (2.15) and (2.18) for each flow component is

$$u_{ij}^{\mathrm{n}} \delta_x u_{ij}^{\mathrm{n}} + v_{ij}^{\mathrm{n}} \delta_y u_{ij}^{\mathrm{n}} + \frac{u_{ij}^{\mathrm{n+1}} - u_{ij}^{\mathrm{n}}}{\Delta t} = 0 \tag{2.29}$$

$$u_{ij}^{\mathrm{n}} \delta_x v_{ij}^{\mathrm{n}} + v_{ij}^{\mathrm{n}} \delta_y v_{ij}^{\mathrm{n}} + \frac{v_{ij}^{\mathrm{n+1}} - v_{ij}^{\mathrm{n}}}{\Delta t} = 0 \tag{2.30}$$

where operators δ_x and δ_y denote finite difference operators applied in the x and y axes and are defined in Table 2.2. The unknowns in Equations (2.29) and (2.30) are the flow components $(u_{ij}^{\mathrm{n+1}}, v_{ij}^{\mathrm{n+1}})$ for the next time step $n+1$.

	$\delta_x c_{ij}$	$\delta_y c_{ij}$
backward	$\delta_{x\text{-}} = c_i - c_{i-1}$	$\delta_{y\text{-}} = c_j - c_{j-1}$
central	$\delta_{x0} = c_{i+1} - c_{i-1}$	$\delta_{y0} = c_{j+1} - c_{j-1}$
forward	$\delta_{x+} = c_{i+1} - c_i$	$\delta_{y+} = c_{j+1} - c_j$

Table 2.2: Finite difference operators in the x and y axes for a scalar field c.

Similarly, the discrete version of the scalar field transport in Equation (2.14) is

$$\frac{c_{ij}^{\mathrm{n+1}} - c_{ij}^{\mathrm{n}}}{\Delta t} + u_{ij}^{\mathrm{n}} \delta_x c_{ij}^{\mathrm{n}} + v_{ij}^{\mathrm{n}} \delta_y c_{ij}^{\mathrm{n}} = 0 \tag{2.31}$$

where $c_{ij}^{\mathrm{n+1}}$ is the unknown state of c at time $n+1$. Equation (2.31) is used for the numeric propagation of the image parameter ${}^{\mathrm{h}}\hat{Y}$ and each component of ${}^{\mathrm{h}}{}_{\Delta}\mathbf{\Phi}$ in Equations (2.17) and (2.16), respectively. Although these equations are linear with respect to the optical flow and hence, easier to implement numerically, it is preferred to use the same numerical scheme used for Equations (2.29) and (2.30) to maintain regularity in the implementation.

2.5.2 Upwind finite differences method

Direct application of any difference operator in Table 2.2 without proper consideration of the underlying problem either will give unsatisfactory results or will be unstable. For example, using central difference operators δ_{x0} and δ_{y0} in Equations (2.29) and (2.30) will result in unstable results after a few iterations.

Within the finite difference methods, the *upwind finite difference methods* offer a simple solution to this instability problem. Consider an optical flow vector such as the one in Figure 2.5. The upwind region relative to this flow vector is on the left and bottom sides of $\mathbf{p}$. This region has already been "visited" by the optical flow and its information content is known to it. Consequently, it is safe to use backward difference operators $\delta_{x\text{-}}$ and $\delta_{y\text{-}}$ for the computation of discrete derivatives in Equation (2.31).

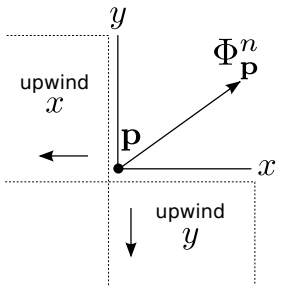


Figure 2.5: Upwind direction with respect to optical flow.

However, direct application of the same finite difference method to the non-linear Equations (2.29) and (2.30) will not generate satisfactory results (see Thomas book [Thomas, 1999, p. 140] for an example). Instead of using a non-linear PDE solver, which would increase the computational cost excessively, a simple linearization of Equations (2.29) and (2.30) is

	u_{ij}	v_{ij}
backward	$\sigma_{x\text{-}} u_{ij} = u_{i-1,j}$	$\sigma_{y\text{-}} v_{ij} = v_{i,j-1}$
forward	$\sigma_{x+} u_{ij} = u_{i+1,j}$	$\sigma_{y+} v_{ij} = v_{i,j+1}$

Table 2.3: Shift operators.

proposed, enabling the use of finite difference methods. For this, the term *dominant flow* is introduced to refer to the largest flow vector in the pixel's 4-neighborhood. It is calculated as

$$\hat{u}_{ij}^{\mathtt{n}} = \begin{cases} \sigma_{x+} u_{ij}^{\mathtt{n}} & \text{if } \delta_{x0} |u_{ij}^{\mathtt{n}}| > 0 \\ \sigma_{x\text{-}} u_{ij}^{\mathtt{n}} & \text{if } \delta_{x0} |u_{ij}^{\mathtt{n}}| \leq 0 \end{cases} \tag{2.32}$$

$$\hat{v}_{ij}^{\mathtt{n}} = \begin{cases} \sigma_{y+} v_{ij}^{\mathtt{n}} & \text{if } \delta_{y0} |v_{ij}^{\mathtt{n}}| > 0 \\ \sigma_{y\text{-}} v_{ij}^{\mathtt{n}} & \text{if } \delta_{y0} |v_{ij}^{\mathtt{n}}| \leq 0 \end{cases} \tag{2.33}$$

where σ denote shift operators in the image grid. These operators are defined in Table 2.3.

The dominant flow vector is important at regions in the image with large flow discontinuities, such as object boundaries. In these regions, foreground objects occlude part of the background. For pixels right on the background side of the flow discontinuity, it is reasonable to use the dominant flow to propagate pixels in the boundary, as it will properly propagate the foreground optical flow discontinuity.

2.5.3 Iterative numerical scheme

The time iterations of the upwind numerical scheme for the propagation equations is as follows. First, the dominant flow $\hat{u}^{\mathtt{n}}$ is computed for all pixels in the image grid using Equation (2.32). Then, the optical flow and scalar fields are propagated in the x direction as

$$u_{ij}^{\mathtt{n}+\frac{1}{2}} = \begin{cases} u_{ij}^{\mathtt{n}} - R\hat{u}_{ij}^{\mathtt{n}} \delta_{x\text{-}} u_{ij}^{\mathtt{n}} & \text{if } \hat{u}_{ij}^{\mathtt{n}} \geq 0 \\ u_{ij}^{\mathtt{n}} - R\hat{u}_{ij}^{\mathtt{n}} \delta_{x+} u_{ij}^{\mathtt{n}} & \text{if } \hat{u}_{ij}^{\mathtt{n}} < 0 \end{cases} \tag{2.34}$$

$$v_{ij}^{\mathtt{n}+\frac{1}{2}} = \begin{cases} v_{ij}^{\mathtt{n}} - R\hat{u}_{ij}^{\mathtt{n}} \delta_{x\text{-}} v_{ij}^{\mathtt{n}} & \text{if } \hat{u}_{ij}^{\mathtt{n}} \geq 0 \\ v_{ij}^{\mathtt{n}} - R\hat{u}_{ij}^{\mathtt{n}} \delta_{x+} v_{ij}^{\mathtt{n}} & \text{if } \hat{u}_{ij}^{\mathtt{n}} < 0 \end{cases} \tag{2.35}$$

$$c_{ij}^{\mathtt{n}+\frac{1}{2}} = \begin{cases} c_{ij}^{\mathtt{n}} - R\hat{u}_{ij}^{\mathtt{n}} \delta_{x\text{-}} c_{ij}^{\mathtt{n}} & \text{if } \hat{u}_{ij}^{\mathtt{n}} \geq 0 \\ c_{ij}^{\mathtt{n}} - R\hat{u}_{ij}^{\mathtt{n}} \delta_{x+} c_{ij}^{\mathtt{n}} & \text{if } \hat{u}_{ij}^{\mathtt{n}} < 0 \end{cases} \tag{2.36}$$

After this, dominant flow $\hat{v}^{\mathtt{n}+\frac{1}{2}}$ is computed applying Equation (2.33) $v_{ij}^{\mathtt{n}+\frac{1}{2}}$. Then, the

propagation takes place in the y direction

$$u_{ij}^{n+1} = \begin{cases} u_{ij}^{n+\frac{1}{2}} - R\hat{v}_{ij}^{n+\frac{1}{2}}\delta_{y-}u_{ij}^{n+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{n+\frac{1}{2}} \geq 0 \\ u_{ij}^{n+\frac{1}{2}} - R\hat{v}_{ij}^{n+\frac{1}{2}}\delta_{y+}u_{ij}^{n+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{n+\frac{1}{2}} < 0 \end{cases} \tag{2.37}$$

$$v_{ij}^{n+1} = \begin{cases} v_{ij}^{n+\frac{1}{2}} - R\hat{v}_{ij}^{n+\frac{1}{2}}\delta_{y-}v_{ij}^{n+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{n+\frac{1}{2}} \geq 0 \\ v_{ij}^{n+\frac{1}{2}} - R\hat{v}_{ij}^{n+\frac{1}{2}}\delta_{y+}v_{ij}^{n+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{n+\frac{1}{2}} < 0 \end{cases} \tag{2.38}$$

$$c_{ij}^{n+1} = \begin{cases} c_{ij}^{n+\frac{1}{2}} - R\hat{v}_{ij}^{n+\frac{1}{2}}\delta_{y-}c_{ij}^{n+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{n+\frac{1}{2}} \geq 0 \\ c_{ij}^{n+\frac{1}{2}} - R\hat{v}_{ij}^{n+\frac{1}{2}}\delta_{y+}c_{ij}^{n+\frac{1}{2}} & \text{if } \hat{v}_{ij}^{n+\frac{1}{2}} < 0 \end{cases} \tag{2.39}$$

The constant term

$$R := \frac{\Delta t}{\Delta x} = \frac{1}{N} \tag{2.40}$$

is the ratio between time step and pixel separation.

The stability of the numerical scheme is given by the Courant-Friedrichs-Lewy condition Thomas [1995]

$$R \max_{n,i,j} \left\{ |\hat{u}_{ij}^{n}|, |\hat{v}_{ij}^{n}| \right\} \leq 1 \tag{2.41}$$

Given the number of iterations N as a parameter of the algorithm, it is easy to see that the components of the optical flow field must satisfy inequality $|u_{ij}^{n}|, |v_{ij}^{n}| \leq N$ for all pixels and time iterations. Considering that the filter pyramid is constructed with a down-sampling factor of 2 and thus, the optical flow is scaled by one-half at each higher level, level $h+1$ requires only half the number of iterations as level h.

2.5.4 Numerical artifacts

Figures 2.7 and 2.8 show the results of propagating an image and a ground truth optical flow field from the 'Bunny' dataset. The scheme is configured with $N = 3$ iterations between image time steps, and runs for 50 image time steps, that is, $50 * N$ numeric iterations. The maximum optical flow in this sequence is around 2.7 pixels and is located near the head of the bunny.

Errors in the predicted image and optical flow are small for the first 10 images, and starts increasing at flow discontinuities. Notice that regions with higher errors are located on the right-hand side of the bunny. In these areas, the scene's background plane is being discovered, as the Bunny moves to the left of the image, leaving a trail of blurred optical flow. In the *leading edge* of the bunny, that is, the flow discontinuity on the left side of the object, the sharpness of the flow field is preserved even after 50 image steps.

This difference between errors in the *leading edge* and the *trailing edge* of optical flow discontinuities has a physical explanation by considering the type of flow Equation (2.12) is modeling. Consider for example a wave in the beach. Before it breaks, the wave approaches as a wall of water (like in *Interstellar* movie). This wall of water is a shock wave and its steepness remains as the wave moves forward. Once the wave passes, one will notice that the back side of the wave has a smooth transition to normal sea level (like Figure 2.6. This smooth transition preserves the entropy added to the system by the moving wave. Any other shape in the trailing edge of the wave will be some sort of ordered transition that reduces the

entropy of the system, going against the second law of thermodynamics. In fluid dynamics, the optical flow conservation equation derived in Equation (2.12) is known as the inviscid Burgers' Equation, and models the evolution of an inviscid fluid in space. Like any other real physical phenomenon, it obeys the laws of thermodynamics. The upwind finite difference solver used to propagate the optical flow field naturally includes the conservation of entropy within the numerical scheme.

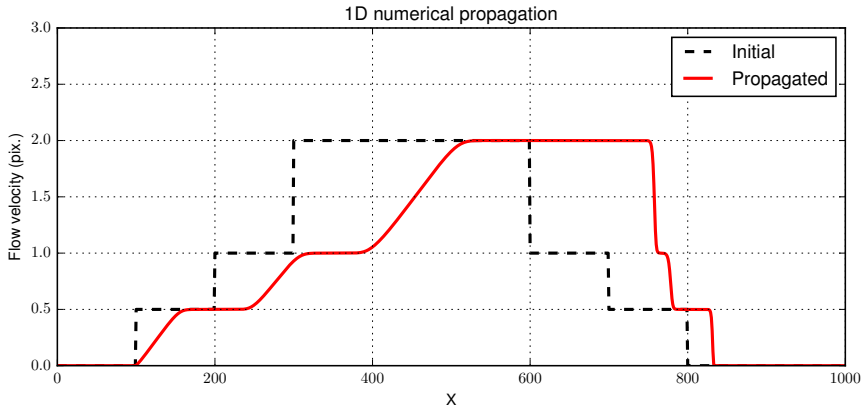


Figure 2.6: One-dimensional optical flow propagation. Discontinuities in the leading edges (right) are preserved during the propagation. The trailing edges (left) are naturally smoothed by the numerical scheme.

Optical flow, is not a physical flow field, and discontinuities on both leading and trailing edges have valid interpretations explained as motion boundaries typically originated at object boundaries. It is desirable to consider a relaxation on the physical interpretation of the flow that could be included explicitly in the numerical method to obtain a more accurate propagation result. Such a task is, however, beyond the scope of this thesis and will be left as future work.

2.5.5 Some comments about GPU and FPGA implementations

The numerical solution to the propagation equations poses several properties that can be exploited in a parallel or streamed implementation. The propagation of either the optical flow or the associated scalar fields only require data in the 4-neighborhood of each pixel. This is ideal for a GPU implementation as one can exploit the data locality of pixels stored in global memory to accelerate their read.

For a stream-type implementation, as I would implement this scheme on a FPGA, one will only need to buffer 3 rows of data for each half time step propagation. x and y propagation schemes can be connected in cascade inside the FPGA, and each block only reads the output of the block immediately before. Thus, the FPGA implementation will need to read the initial conditions in raster order only once from external memory and write, also in raster order, the propagated output. Such a design would maximize pixel throughput and minimize the need to access external memory.

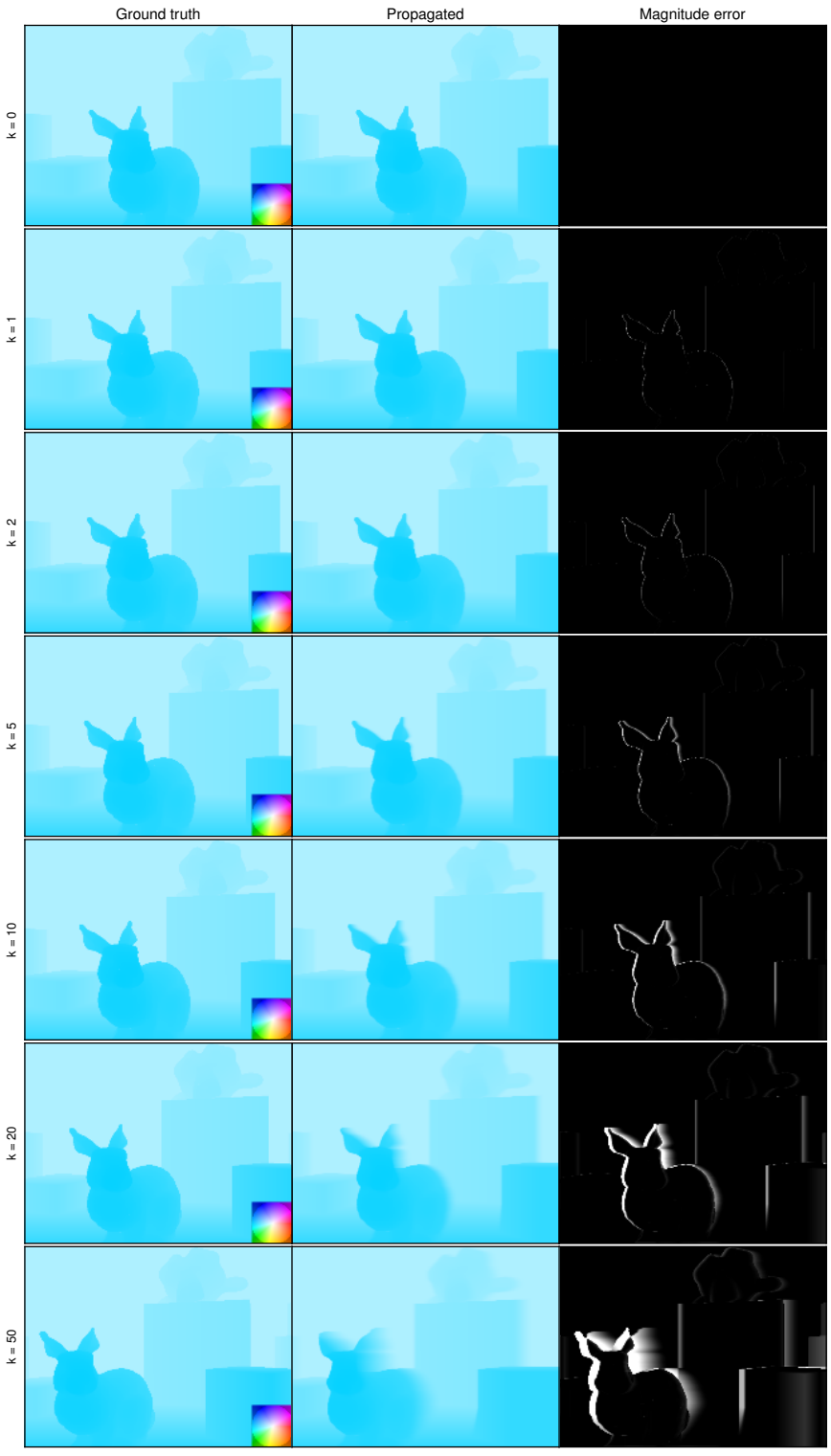


Figure 2.7: Numerical propagation of optical flow. Left: ground truth field, middle: propagated flow, right: magnitude error in range $[0, 1]$ pixels.

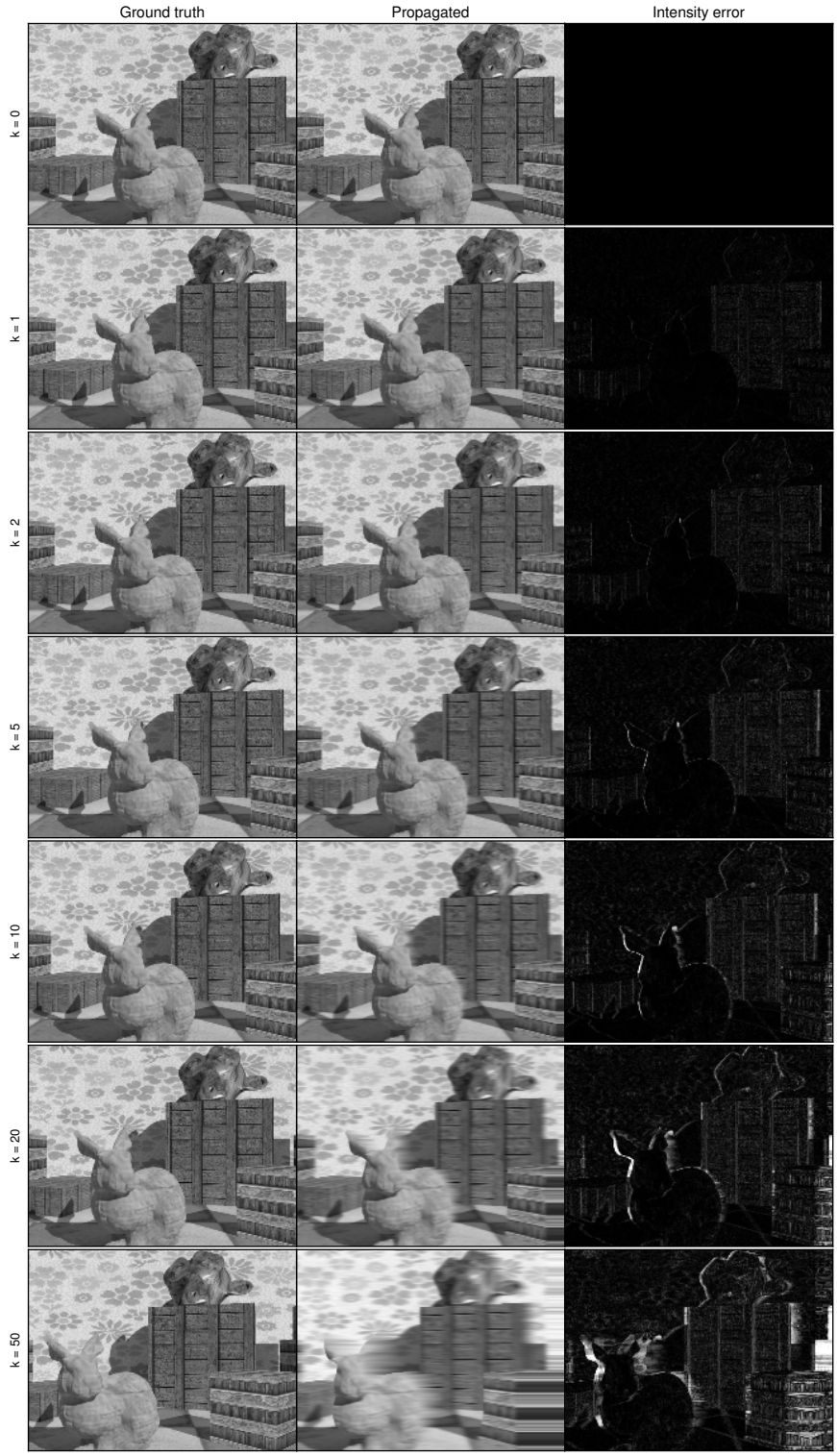


Figure 2.8: Numerical propagation of image field by the optical flow. Left: ground truth image, middle: propagated image, right: brightness error $[0, 50\%]$.

2.6 Experimental Results

This section presents the experimental evaluation of the optical flow filter algorithm. The evaluation is divided in two components. First, the optical flow filter is compared against other open source algorithms on ground truth image sequences. Second, qualitative results are presented for real-life high-speed image sequences captured while driving a car vehicle at ANU campus and Canberra.

In the evaluation, optical flow algorithms capable of running at real-time frame rates are used. The algorithms are: the pyramidal implementation of Lucas-Kanade method Bouguet [2001], Farneback algorithm Farnebäck [2003] and Brox *et. al.* total variational method Brox et al. [2004]. The implementation of these algorithms is available in the GPU library module of OpenCV. The Massively Parallel Lucas-Kanade method by Plyer *et. al.* Plyer et al. [2014] could not be included in the evaluation as there is no source code or executable available to test the algorithm. The algorithm by Werlberger Werlberger [2012] could not be tested as his test application requires a legacy version of Nvidia CUDA.

2.6.1 Ground truth optical flow dataset

To analyze the error performance of the optical flow filtering algorithm, one needs an image dataset that replicates the operating conditions in which the algorithm is expected to be deployed. That is, for the purpose of this thesis, a dataset should provide images from a high-speed camera moving in complex environments. Moreover, the length of the dataset should be enough for the filter algorithm to converge to a dense optical flow field.

These two properties are not present on any of the standard datasets available for optical flow evaluation. Those are: the Middlebury dataset by Baker et al. [2011], the Kitti dataset by Geiger et al. [2013] and the synthetic Sintel dataset by Butler et al. [2012]. These datasets either provide short image sequences for which the optical flow filter cannot converge (such is the case of the Middlebury and Kitti datasets) or the pixel displacement between frames is too large for the algorithm to be able to identify.

Considering this, a synthetic dataset using Blender 3D modeling software[10] is created to simulate a high-speed camera moving in a complex environment. The dataset, named Bunny, is shown in Figure 2.9 as a panoramic view. The camera moves to the right with a constant linear velocity of $0.5\,\mathrm{ms}^{-1}$ during one second, thus generating a total of 300 images. The ground truth optical flow is calculated from the rendered depth math using the camera velocity and intrinsic parameters, as explained in Appendix A.

[10] https://www.blender.org/

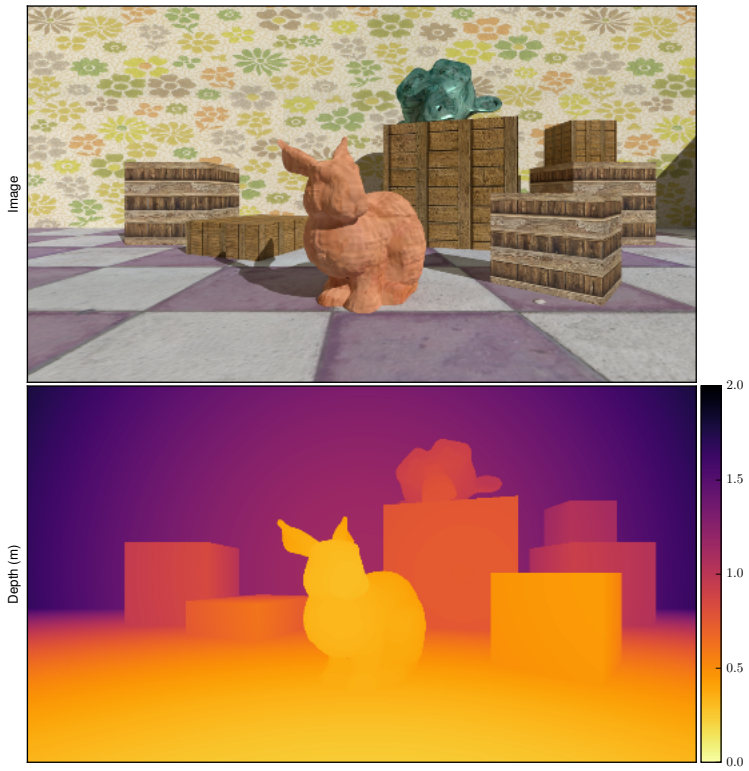


Figure 2.9: Panoramic view of Bunny 3D environment. Ground truth optical flow is computed using camera motion and intrinsic parameters and the rendered depth map.

2.6.2 Error metrics

The standard error metrics proposed in the Middlebury dataset are used. Those are: *Endpoint Error* (EE) and *Angular Error* (AE). The Endpoint Error is the magnitude of the difference between the ground-truth optical flow and the algorithm's estimation. That is

$$\mathrm{EE} = \sqrt{(u - u_{\mathrm{gt}})^2 + (v - v_{\mathrm{gt}})^2} \tag{2.42}$$

The Angular Error (AE) Baker et al. [2011] measures the angular error between estimated and ground truth optical flow using 3D vectors $(1, u, v)$ and $(1, u_{\mathrm{gt}}, v_{\mathrm{gt}})$. It is calculated as

$$\mathrm{AE} = \arccos\left(\frac{1 + u \cdot u_{\mathrm{gt}} + v \cdot v_{\mathrm{gt}}}{\sqrt{1 + u^2 + v^2}\sqrt{1 + u_{\mathrm{gt}}^2 + v_{\mathrm{gt}}^2}}\right) \tag{2.43}$$

The angular error includes the constant 1 to seamlessly handle the case of zero flow vectors.

2.6.3 Evaluation on the Bunny sequence

Figure 2.10 shows the estimated optical flow for the flow-filter algorithm as well as its end-point error field. Figure 2.11 shows the optical flow for all the evaluated algorithms. Table

2.4 enumerates the parameters used for each algorithm. These parameters were selected trying to run each of the algorithms as fast as possible to match the frame rate of the camera. An important observation is that the parameters for Brox algorithm are set such that the algorithm computes accurate optical flow fields, regardless of runtime. This is done to have a reference of high-accuracy flow estimates. Table 2.5 summarizes the error metrics of the evaluated algorithms.

Regarding the optical flow filter algorithm in Figure 2.10, the initial condition of the filter state is set to zero and flow starts to be identified at brightness discontinuities ($k = 1, 2$). The estimated flow spreads to textureless regions as the camera moves and the smoothing filter is applied to the updated flow. This can be observed at the interior of the floor tiles ($k = 5$). After approximately 25 frames (or 0.08 seconds) the optical flow covers almost all regions of the image. At this point, the filter state maintains a dense estimate constantly updated given new image data. Last three rows of Figure 2.10 illustrates the trailing edge artifact of the filter algorithm. Optical flow in the leading edge (left side of the bunny) shows a sharp transition between background and foreground flow. At the trailing edge (right hand side), background objects are being discovered and the optical flow goes from high to low. This transition is, however, not immediate and needs several time steps for the algorithm to "forget" about the foreground flow. The width of the trailing edge is related to parameter γ in the filter update stage (Section 2.4.3).

Flow filter	**Value**
levels (H)	2
${}^{h}\gamma$	$[50, 5]$
max flow	4
smooth iterations	$[2, 4]$

Pyramidal Lucas-Kanade	**Value**
levels	2
window width	5
window height	5
iterations	5

Farneback	**Value**
levels	2
pyramid scale	0.5
window size	3
iterations	2
polynomial N	5
sigma	1.1
fast pyramids	yes

Brox	**Value**
levels	2
pyramid scale	0.5
alpha	0.197
gamma	50
solver iterations	50
inner iterations	100

Table 2.4: Parameters of evaluated algorithms in the Bunny sequence.

2.6.3.1 Runtime vs. error performance

A second study to analyze the relationship between runtime and error performance for the evaluated algorithms is performed. This analysis helps to give an idea where the flow-filter algorithm sits with respect to other slow but more accurate methods.

For this analysis, the optical flow estimated by the flow-filter at frame $k = 150$ is chosen as reference for the comparison with the other algorithms. For the remaining algorithms,

	Flow-filter	Pyr-LK	Farneback	Brox
avg. EE (pix.)	0.127	0.349	0.249	0.113
avg. AE (deg.)	2.498	7.094	5.740	1.799
avg. runtime (ms)	1.227	2.037	2.508	1185.963
avg. frame rate (Hz)	814.499	490.791	398.707	0.843
avg. throughput (Mpix/s)	250.214	150.771	122.482	0.259

Table 2.5: Results summary for the Bunny sequence.

the number of iterations (inner-iterations in Brox method) is varied between 0 and 100, and the optical flow, runtime and endpoint error are recorded. Figure 2.12 plots the relationships between the number of iterations and the average endpoint error (second row), the resulting frame rate for given number of iterations (third row) and the correlation between frame rate and the average endpoint error (fourth row).

Brox total variational methods gets the most benefit from incrementing the number iterations, reaching similar average endpoint at around 30 iterations as the flow-filter algorithm. This level of accuracy, as one would expect, decreases the frame rate, reaching approximately 30 Hz, compared to more than 800 Hz for the flow-filter. Lucas-Kanade and Farneback algorithms show no significant difference in error performance, however the frame rate of both algorithms drastically reduces as the number of iterations increases.

2.6.4 Evaluation on the Middlebury test dataset

The Middlebury test sequences are used to provide an extra level of comparison for the optical flow filter algorithm. This evaluation allows to compare the error metrics of the flow-filter algorithm with other algorithms in the literature.

The original test sequence consists of only 5 images and one ground truth optical flow field. As mentioned before, the flow-filter algorithm requires several frames to reach a dense flow field estimation. Given the ground truth optical flow, it is possible to enlarge the dataset by interpolating the images between two frames. To achieve this, the ground truth provided by the dataset is used to propagate the input image 50 frames backwards and forwards using a similar approach as described in Section 2.5.4. This creates a total of 100 images which is sufficient for the flow-filter to converge to a meaningful flow estimate. The estimated optical flow at time $k = 100$ is scales by 50 to compare it with the ground truth. These images are used by the flow-filter and the output at time $k = 100$ is scaled by 50 to compare it with the ground truth.

For comparison purposes, the parameters of the evaluated algorithms are set to values matching those reported in eFOLKI evaluation Plyer et al. [2014]. Table 2.6 lists the parameters for each algorithm. Figure 2.14 shows the optical flow for each of the evaluated algorithms and Figure 2.13 shows the estimated optical flow and endpoint error for the flow-filter algorithm. Tables 2.7 and 2.8 summarize the endpoint and angular error error metrics for the evaluated algorithms, respectively. For the angular error, the values reported in the original eFOLKI paper are included for comparison.

In general, the flow-filter algorithm outperforms the pyramidal Lucas-Kanade method both

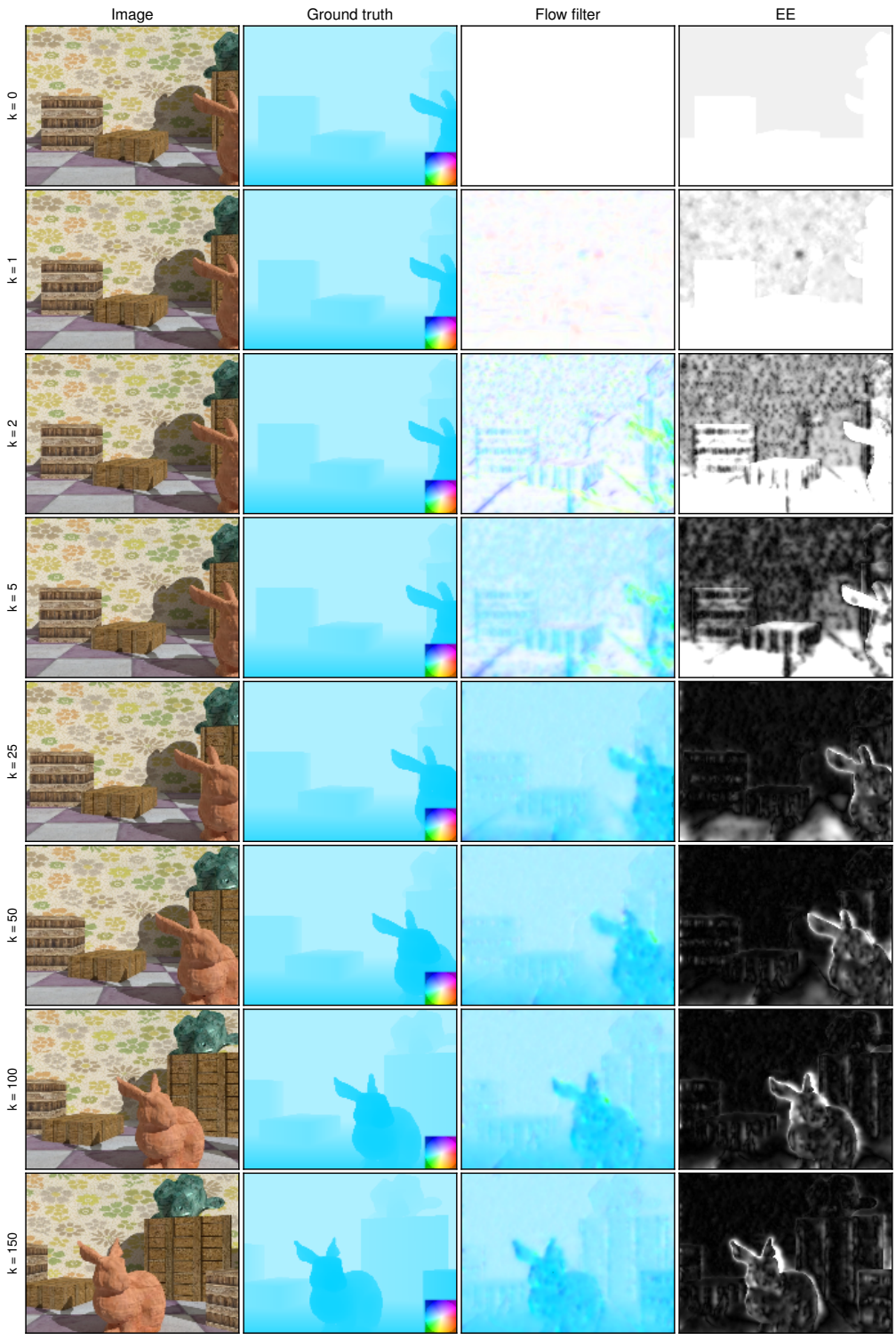


Figure 2.10: Flow-filter algorithm results on the Bunny sequence.

Figure 2.11: Estimated optical flow on the Bunny sequence.

in endpoint and angular error, and reports comparable values to eFOLKI method. More importantly, the flow-filter algorithm gives 1.14 ms average runtime per image frame (or 870 Hz), which is 5 times smaller than the runtime recorded for eFOLKI (6 ms, 166 Hz). This translates to almost 200 Mpix/s difference in throughput (244 Mpix/s for flow-filter compared to 51.2 Mpix/s for eFOLKI).

2.6.5 Evaluation on real-life high-speed video

The last set of evaluations on the optical flow filter algorithm are performed using real-life high-speed video data. Video data was recorded while driving a vehicle around ANU campus

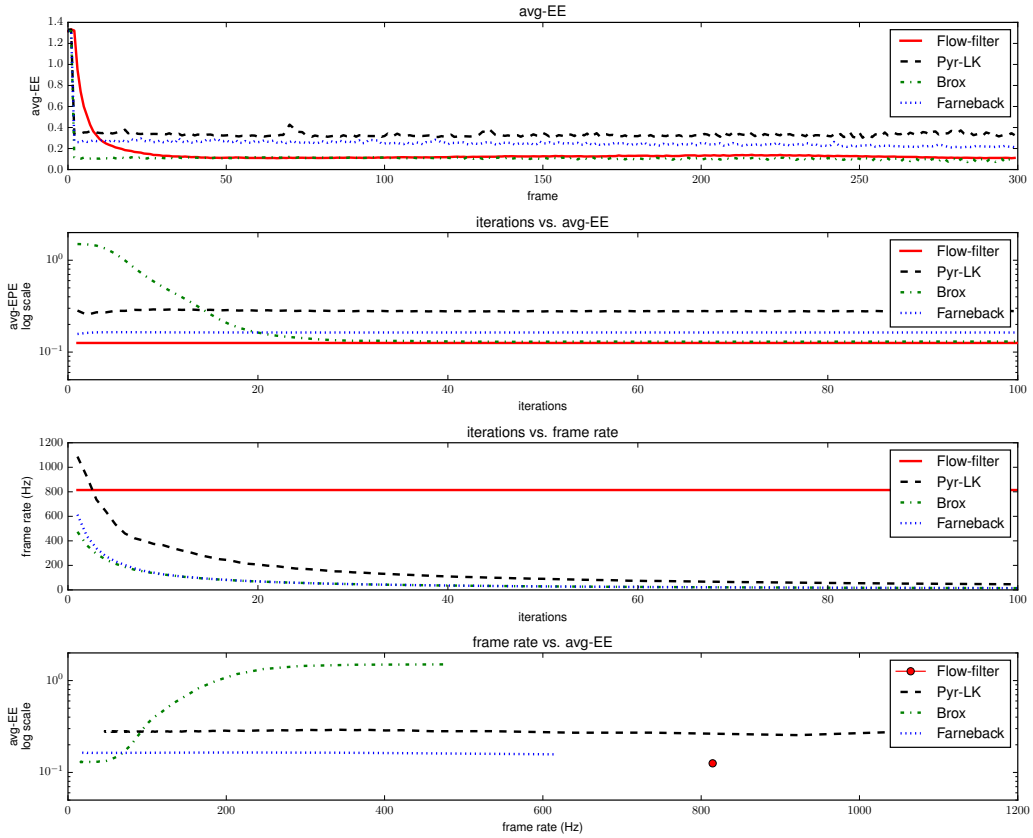


Figure 2.12: Trade-off between accuracy and runtime performance in the Bunny sequence.

Flow filter	**Value**
levels (H)	2
${}^{h}\gamma$	$[50, 5]$
max flow	4
smooth iterations	$[2, 4]$

Pyramidal Lucas-Kanade	**Value**
levels	2
window width	17
window height	17
iterations	20

Farneback	**Value**
levels	2
pyramid scale	0.5
window size	17
iterations	20
polynomial N	5
sigma	1.1
fast pyramids	yes

Brox	**Value**
levels	10
pyramid scale	0.5
alpha	0.54
gamma	450
solver iterations	100
inner iterations	100

Table 2.6: Parameters of evaluated algorithms in the Middlebury test sequences.

and in Civic area in Canberra, Australia. The camera used is a Basler ACE acA2000-165um monochrome USB3 camera. This camera is configured to record 1016×544 images at 300Hz.

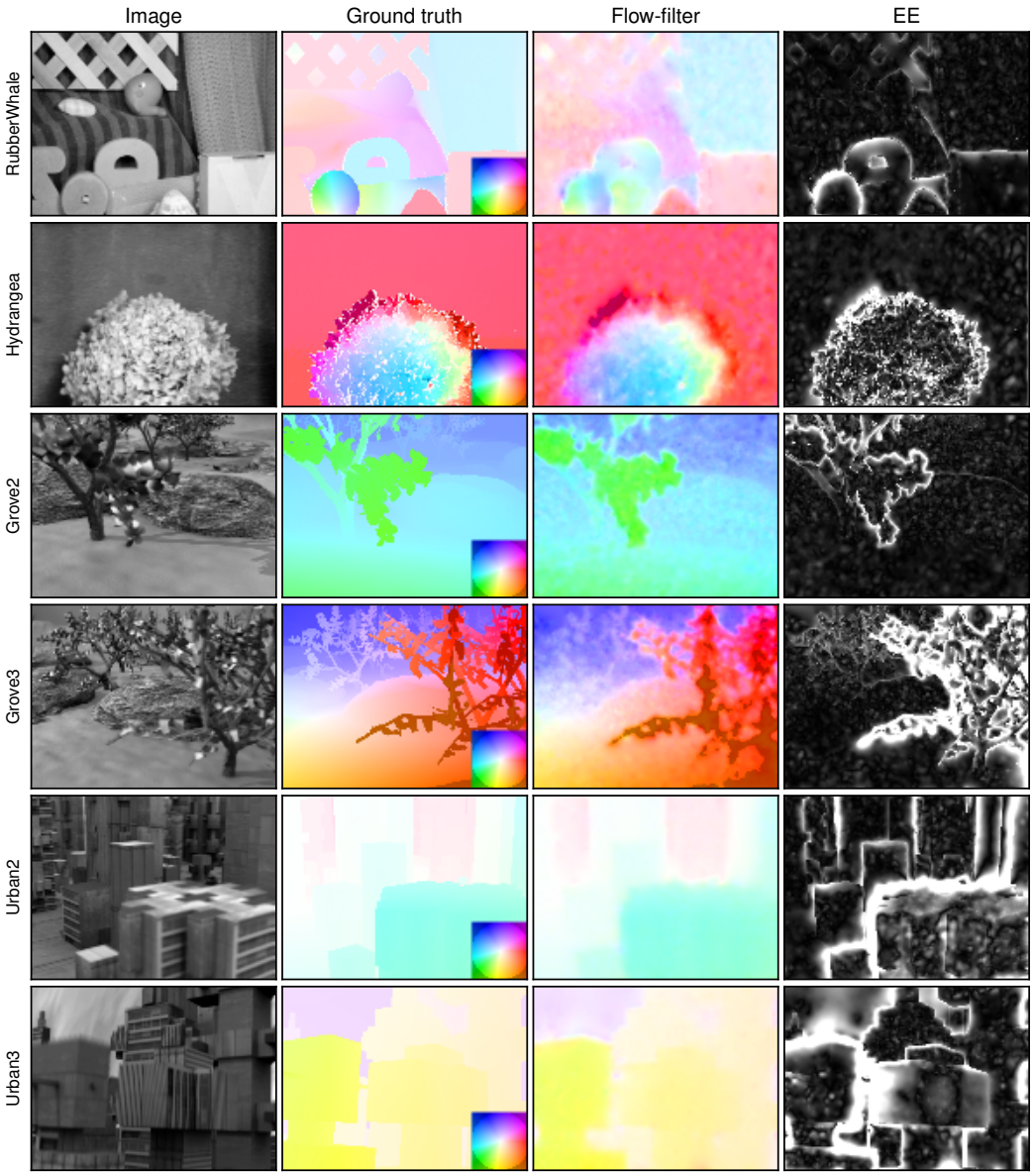


Figure 2.13: Results of flow-filter algorithm in the interpolated Middlebury test dataset.

Table 2.9 enumerates the camera's most relevant configuration parameters.

Image data was processed offline to record the optical flow and runtime. The flow-filter algorithm was configured using the parameters listed in Table 2.10. Figure 2.16 shows some examples of scenes and optical flow computed from video captured at the ANU campus. Figure 2.17 shows a comparison between the flow-filter and Pyr-LK algorithms on the same video scenes. With no ground truth available for these sequences, it is only possible to offer a qualitative evaluation of the results.

On average, the reported runtime of the flow-filter algorithm is 1.62 ms per image frame. This translates to frame rates of approximately 616 Hz. In contrast, the Pyramidal Lucas-

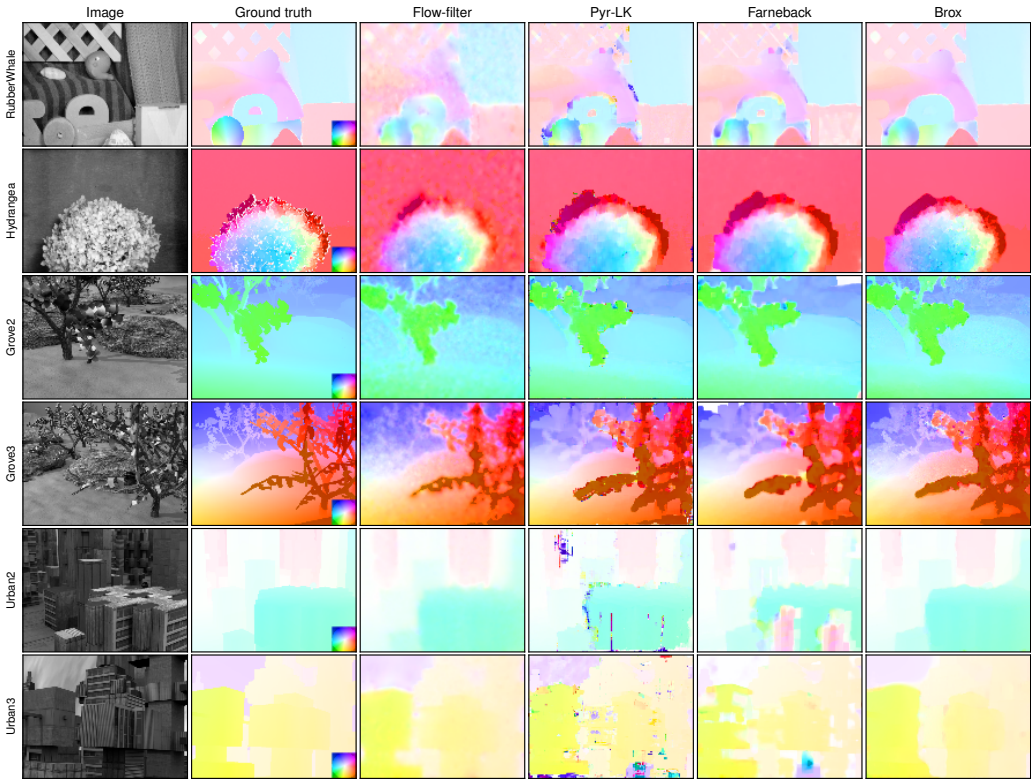


Figure 2.14: Evaluated algorithms in the Middlebury test dataset.

avg. EE (pix.)	**Flow-filter**	**Pyr-LK**	**Farneback**	**Brox**
RubberWhale	0.317	0.323	0.310	0.180
Hydrangea	0.584	0.786	0.597	0.467
Grove2	0.345	0.395	0.383	0.297
Grove3	0.920	1.368	1.162	0.921
Urban2	0.886	2.838	3.889	0.497
Urban3	0.999	3.293	2.343	1.143
avg. runtime (ms)	**1.140**	**33.513**	**27.626**	**322.067**
avg. frame rate (Hz)	**876.721**	**30.933**	**37.246**	**3.166**
throughput (Mpix/s)	**244.863**	**8.484**	**10.107**	**0.866**

Table 2.7: Average Endpoint Error results in the Middlebury test dataset.

Kanade method (2 levels, 5×5 window and 6 iterations) yields runtime performance of 3 ms per image frame, or 332.3 Hz frequency. While Pyr-LK reaches the same frame rate as the camera recording frequency, the output optical flow presents high levels of noise and pixels with invalid flow estimates. Extra post-processing is required to sanitize the output field and extract useful information.

Currently, there are two publicly available videos in Youtube displaying the results using the layout of Figure 2.15. The URLs for these videos are listed in Table 2.11.

avg. AE (deg.)	Flow-filter	Pyr-LK	Farneback	Brox	eFOLKI
RubberWhale	9.8	10.2	10.4	5.6	8.6
Hydrangea	7.9	8.0	7.8	7.0	3.6
Grove2	4.7	5.8	5.8	4.1	4.7
Grove3	8.5	12.1	10.7	8.4	9.8
Urban2	7.0	10.9	20.2	3.7	7.7
Urban3	6.4	16.0	19.4	9.8	14.8
avg. runtime (ms)	**1.140**	**33.513**	**27.626**	**322.067**	**6**
avg. frame rate (Hz)	**876.721**	**30.933**	**37.246**	**3.166**	**166.667**
throughput (Mpix/s)	**244.863**	**8.484**	**10.107**	**0.866**	**51.2**

Table 2.8: Average Angular Error results in the Middlebury test dataset. Results for eFOLKI were taken from the original article of Plyer et al. [2014].

Parameter	Value
Resolution	1016×544
Horizontal binning	2
Vertical Skipping	2
Exposure time	30 us
Frame rate	300 Hz
Trigger	Software

Table 2.9: Basler USB3 camera parameters.

2.6.6 Runtime performance on embedded GPU hardware

Table 2.12 shows a comparison of the runtime of the flow-filter algorithm running on the Nvidia Tegra K1 embedded System on Chip and the GTX 780 desktop GPU card. The GPU in the Tegra K1 SoC has 192 CUDA cores with Kepler architecture and approximately 12 GB/s shared memory bandwidth[11]. For reference, the GTX 780 has 2304 CUDA cores with Kepler architecture and 288 GB/s dedicated memory bandwidth[12].

The flow-filter algorithm was run on the T-K1 SoC at a quarter VGA resolution (320×240) and varying the maximum flow allowed in the algorithm. This effectively increases the number of iterations of the numerical scheme for the propagation stage of the filter. The number of smooth iterations was set to 1 for all benchmarks.

For 1 pixel maximum flow, the flow-filter algorithm runs at 114 Hz on the T-K1 chip, and frequency decreases to 74 Hz for flow values up to 8 pixels. It is believed this image resolution and frame rates might be sufficient to provide visual cues to a small robotic vehicle, such as quadrotor, equipped with an embedded computer. Robotic vehicles capable of carrying heavier payloads, such a car, can take advantage of desktop computer components to run the optical flow algorithm at much higher frame rates.

[11] http://www.nvidia.com/object/tegra-k1-processor.html
[12] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-780/specifications

Flow filter	**Value**
levels (H)	2
${}^{\mathrm{h}}\gamma$	$[250, 100]$
max flow	3
smooth iterations	$[2, 3]$

Table 2.10: Flow-filter parameters for real-life high-speed video sequences.

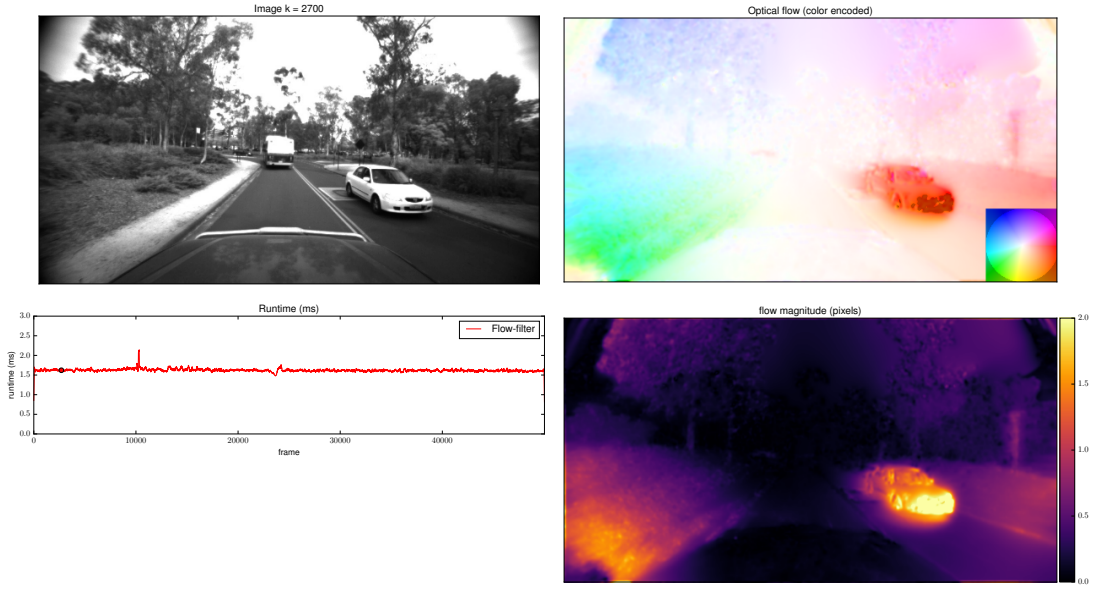


Figure 2.15: Youtube video layout. 300 Hz image sequences were recorded and process offline by the flow-filter algorithm. This video frame shows the current image in gray scale, the estimated optical flow using the color wheel encoding as well as the flow magnitude as a heat map.

Location	**URL**
ANU campus	https://www.youtube.com/watch?v=_oW1vMdBMuY
Canberra - Civic	https://www.youtube.com/watch?v=_Rpi7WS-HTw

Table 2.11: Flow-filter Youtube videos.

2.7 Summary

A filter formulation for computing dense optical flow in real time was introduced. The algorithm is constructed as a pyramid of filter loops, where an optical flow field is incrementally built and constantly refined using new image data and previous state estimation.

Each filter in the pyramid structure is made of an update and propagation stage. The propagation stage takes previous optical flow estimation and propagates it forward in time. This is implemented as a set of partial differential equations modeling the transport of image and optical flow by the optical flow itself. A fast numerical implementation based on final difference methods can efficiently be implemented both on GPU and FPGA hardware.

	T-K1	GTX-780		
Max flow	**320x240**	**320x240**	**640x480**	**1016x544**
1	114 Hz	3382 Hz	1820 Hz	1198 Hz
2	97 Hz	3197 Hz	1496 Hz	973 Hz
4	89 Hz	2660 Hz	1110 Hz	679 Hz
8	74 Hz	1814 Hz	696 Hz	419 Hz

Table 2.12: Runtime performance for embedded Nvidia Tegra K1 SoC and desktop GTX 780 GPU.

In the update stage, the predicted optical flow is corrected using new image data coming from the camera. The formulation of the update follows the standard brightness conservation assumption to create a cost function to recover flow. A temporal regularization terms is added to include the predicted optical flow as a prior solution to this cost at current frame. This makes the problem well defined for all pixels in the image regardless of their texture content. A simple average filter diffuses information from highly textured regions to flat regions, thus covering the whole image with dense flow estimation.

Currently, an open-source implementation of the algorithm is developed for Nvidia GPU hardware using CUDA framework. This implementation is capable of reaching frame rates above 800 Hz working on VGA images.

Experimental evaluation was provided using both ground truth image sequences and real-life high speed video. A synthetic dataset was created using Blender to simulate a high-speed camera moving in a photo-realistic environment. The ground truth optical flow is calculated from the camera's velocity, intrinsic parameters and the rendered depth map. Evaluation on this sequences shows the convergence of the algorithm to dense flow fields after approximately 30 frames, after which the optical flow is maintained with error levels below standard algorithms available for comparison. At the same time, the flow-filter algorithm outperforms the rest of the algorithms in terms of frame rate performance.

A second set of ground truth evaluation experiments were carried out on the Middlebury test dataset of Baker et al. [2011]. To make the evaluation fair, the image sequences were augmented using the ground truth optical flow provided in the dataset. Results of these evaluations show better error performance than some of the state of the art real-time algorithms.

Finally, a qualitative comparison between the flow-filter algorithm and the pyramidal implementation of Lukas-Kanade method (OpenCV) is presented. For this comparison, real-life 300 Hz high-speed video recorded in urban driving scenes at the ANU Campus and Civic are in Canberra was recorded and processed offline. The reader is encouraged to look at the compiled videos publicly available at the URLs in Table 2.11.

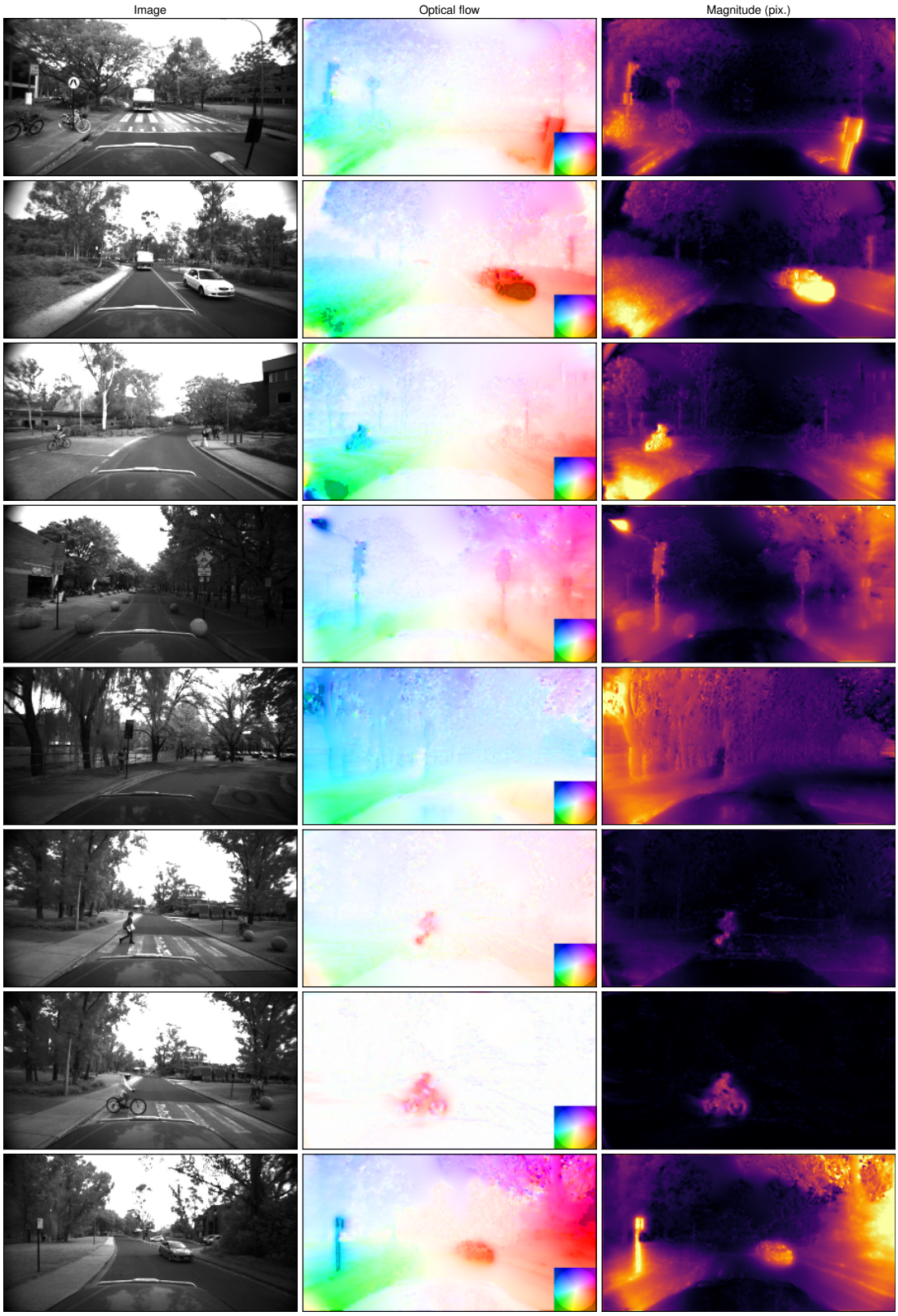


Figure 2.16: ANU campus drive sequence.

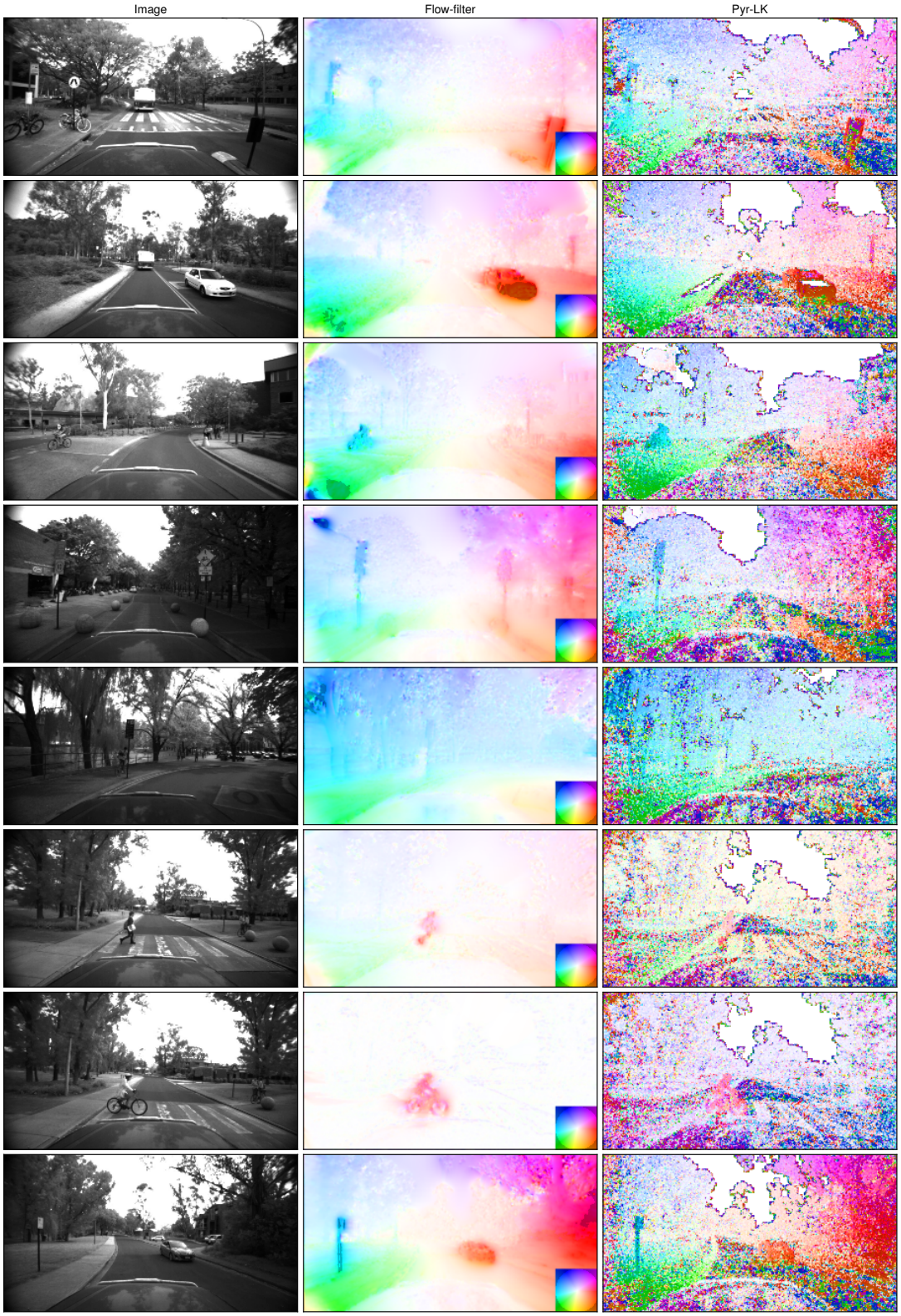


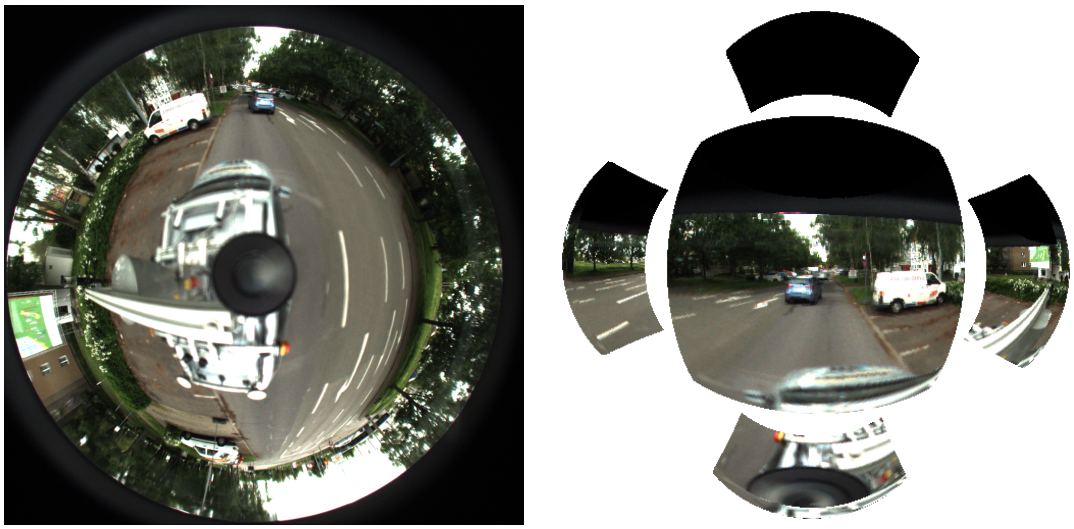
Figure 2.17: ANU campus drive sequence. Comparison between flow-filter and Pyr-LK methods.

Chapter 3

Spherepix: a Data Structure for Spherical Image Processing

One Ring to rule them all.

The Lord of the Rings - J. R. R. Tolkien



(a) Input image.

(b) Spherepix patches.

Figure 3.1: Catadioptric image mapped onto Spherepix patches. Input image from Schönbein et al. [2014].

The use of omnidirectional cameras in robotic applications is growing. This type of camera provides important peripheral vision capabilities to robotic vehicles moving in complex dynamic environments. Omnidirectional cameras use special lenses, such as wide-angle or fish-eye lenses, or a catadioptric system composed of lens and mirrors to project light onto the image sensor, Chahl and Srinivasan [2000]; Geyer and Daniilidis [2001]. Omnidirectional images captured by conventional image sensors such as CCD or CMOS arrays suffer from heavy distortion caused by the camera optics. Consequently, low-level image processing operations

such as linear filtering need to be reformulated to compensate for image distortions. A naive geometric approach to compensating for image distortion in omnidirectional images destroys the shift-invariance and separability of typical linear image processing kernels, leading to a decrease in computational performance.

Images captured by any omnidirectional system can be mapped onto the unit sphere. The sphere offers a regular geometry on which to define image processing operations. Calibration toolbox such as those proposed by Scaramuzza et al. [2006] and Schönbein et al. [2014] can be used to calibrate the optic system in order to ensure geometrically correct images onto the unit sphere. The mapping procedure takes each pixel in the raw (distorted) image and finds its corresponding spherical coordinate. As a consequence, the pixel density in the resulting sphere "pixelation" is not uniform (Figure 3.12d). Consequently, image processing algorithms that rely on pixel correlation on the sphere need to compute relative distance metrics between pixels within the support domain of the algorithm. An example of such approach is the work of Demonceaux et al. [2011] using geodesic metrics to calculate weights for computing image gradient. While this method properly considers the non-uniform sampling of points on the sphere, the filter masks need to be computed for each pixel individually. This contrasts with the simplicity of computing image gradient on standard perspective images using for example Sobel filter masks.

There are two key properties of perspective images that allow efficient implementation of image processing algorithms. First, each pixel (x, y) in the image plane has a direct correspondence to row and column memory coordinates (i, j), enabling fast access of image data. Second, pixels are assumed to be equally separated and orthogonal to each other in the x and y axis. As a result, algorithms for low-level image operations such as Gaussian filtering can exploit the separability of the 2D image coordinate system to create implementations using series of 1D convolutions in x and y, thus decreasing compute time. Considering this, the following fundamental properties are proposed to create an efficient discretization of the sphere on which to apply image processing algorithms:

1. **Local orthonormal pixelation:** That is a pixelation that is both orthogonal and equidistant in the natural geometry of the sphere. This property is fundamental in ensuring shift-invariance (constant coefficient) and separability of convolution kernels.

2. **Efficient memory indexing:** Directions defined in the sphere domain need to map directly to (row, column) coordinates in memory, allowing for direct interpretation of operations applied on these axes as quantities on the sphere.

3. **Independence of camera model:** Algorithms should be defined and implemented on a sphere pixelation, making them invariant to the type of camera used to capture the images.

Properties 1 and 2 are important for creating fast and simple implementations of low-level image operators on the sphere. The aim of property 3 is to abstract out the details of the capturing system and concentrate on creating suitable algorithm implementations on the sphere.

This chapter describes the *Spherepix* data structure Adarve and Mahony [2017] as an approach to implement basic image processing algorithms for spherical representations of omni-

directional images. The aim of the data structure is to create a pixelation of the sphere that approximates the desired properties mentioned above to create fast image processing algorithms. The *Spherepix* framework is the fundamental building block to create a fast implementation of the structure flow algorithm in Chapter 4.

3.1 Outline

The chapter is divided as follows. Section 3.2 reviews the literature regarding spherical pixelations. Section 3.3 introduces relevant geometric definitions used across the chapter. Section 3.4 describes the design process of the Spherepix data structure as well as the relevant pixelation properties. Section 3.5 explains basic image processing algorithms on the pixelation. These includes: camera mapping, linear filtering and manipulation of vector fields. Section 3.6 provides examples on computing SIFT feature points and dense optical flow on the sphere. Finally, Section 3.7 closes the chapter with some summary remarks.

This chapter is also accompanied with Appendix B providing the mathematical details for the regularization procedure applied to the spherepix grids.

3.2 Background

3.2.1 Sphere pixelations

There are several computer data structures used to discretize the sphere. Roughly, these data structures can be classified by the type of surface element used to represent a small portion of the sphere. This section reviews the most relevant sphere pixelation data structures.

The icosahedral subdivision proposed by Baumgardner and Frederickson [1985] uses the regular icosahedron (Platonic solid made of 20 equilateral triangles) as reference for the subdivision. The subdivision splits each triangle into four smaller ones by cutting each edge in the middle point and connecting the new vertex. For efficient storage of the points, it is possible to divide the original icosahedron into ten quadrants made of two triangles each and store the subdivision points of each quadrant as 2D memory arrays. This allows for efficient access of points in the data structure using (row, column) coordinates on each quadrant. To interpolate data within each triangle, one can use barycentric coordinates to compute a weighted average at a given position inside a triangle.

The HEALPix data structure from Górski et al. [2005] uses equal-area four-side polygons to cover the sphere. This data structure is widely used in astronomy for the analysis of celestial data such as microwave background radiation. The HEALPix coverage of the sphere is such that the face elements create isolatitude rings. This property is essential for fast implementation of spherical harmonics. The data structure consists of twelve grid patches: 4 patched in the north pole cap, 4 in the equatorial belt and 4 in the south pole cap. Figure 3.2 illustrates the patch layout. Within each grid, faces can be indexed using a ring indexing scheme where neighbor index correspond to faces in the same isolatitude ring, or nested scheme, where the indexing works as a quad-tree index scheme. Either indexing scheme can be used on the same data structure. The ring scheme is useful for applying Fourier transforms to the spherical data.

Concerning the desired pixelation properties at the beginning of this chapter, HEALPix satisfies properties 2 and 3. Regarding property 1, neighboring points in the pixelation do not form a local orthogonal grid on which one can efficiently run image filters.

The cubic mapping of the sphere, proposed by Ronchi et al. [1996] subdivides the faces of a cube and projects them to the sphere. There are several ways to create the subdivision and mapping: equiangular, conformal, elliptic and gnomic. Each mapping produces pixelations that satisfy different properties such as smooth angular transition between faces, the angle between points or orthogonality of the local coordinate frame Putman and Lin [2007]. Moreover, it is possible to apply a regularization procedure to improve the quality of the pixelation. In their work, the authors use a system of compression and torsion springs to regularize the distance and angle between neighboring points. An important difference between this work and the Spherepix data structure is the fact that Spherepix allows overlapping between cube faces, which further improves the regularity of the generated grid.

The cubic mapping of the sphere provides a simple representation of the sphere in terms of six two-dimensional grids. Indexing of points in the cube data structure is straightforward. This data structure produces a good initial condition for the Spherepix regularization procedure to create equidistant and orthogonal grids.

The last type of sphere data structures relies on overlapping between surface elements to improve pixelation properties. The Yin-Yang by Kageyama and Sato [2004] grid is an example of such data structures. This grid is composed of two surfaces (Yin and Yang) that combined cover the whole sphere, as illustrated in Figure 3.4. The Yin and Yang faces are geometrically identical and consists of rectangular grids sweeping $\pi/2 + \delta$ in latitude and $3\pi/2 + \delta$ in the longitude direction, where δ is a small extra angle to create face overlapping. The subdivision scheme uses equiangular steps in both latitude and longitude directions to create equiangular grids. Operations on each grid can be applied with relative ease. In the border region, one needs to interpolate data from the other grid in order to avoid artificial border artifacts during computations. One consequence of the overlapping is that there are regions of the sphere with two solutions. One will expect that the difference between these solutions is negligible with respect to the discretization step. Alternatively, one can run an extra post-processing algorithm to reconcile the solution in the overlapping area.

One use case example of the Yin-Yang grid is the work of Hara *et. al.* Hara et al. [2015] for the computation of SIFT feature points on the sphere.

3.2.2 Computer vision algorithms on the sphere

There have been many works on implementing classical computer vision algorithms on omnidirectional images. Two cases of particular importance in this thesis are SIFT feature point extraction and optical flow computation.

Regarding optical flow, Daniilidis et al. [2002] define the spatio-temporal derivatives of the Gaussian function on the sphere. Although the derivatives are defined on the sphere, actual computations are performed with the image samples in the original catadioptric image. Bagnato et al. [2009] use local planar coordinates on the sphere's manifold to define discrete operators for spatio-temporal derivatives. The authors use a graph-based representation of the sphere where vertices represent image pixels and edges represent weighted connection between

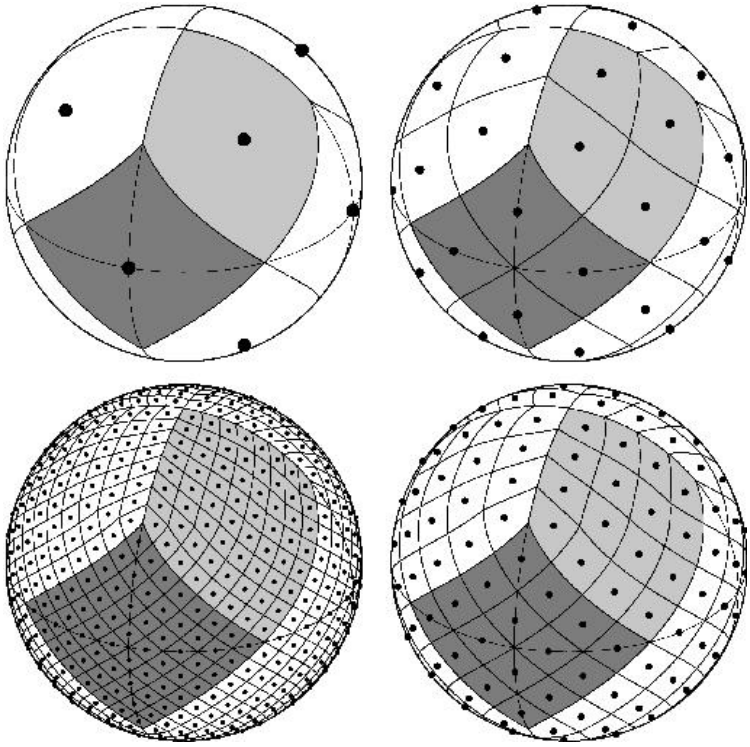
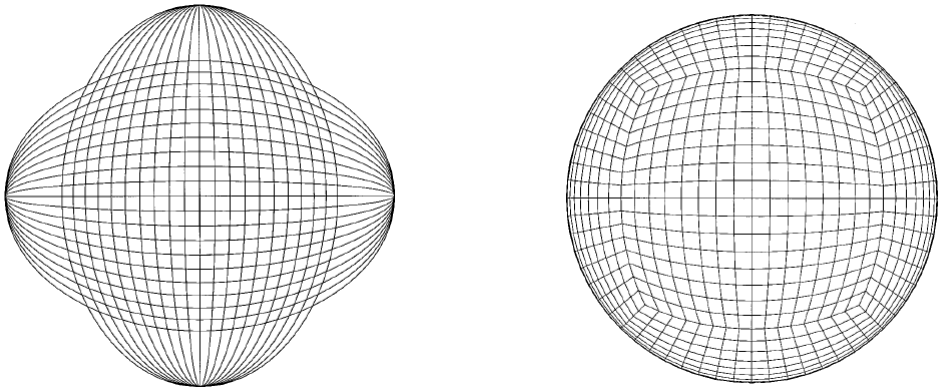


Figure 3.2: HEALPix by *et. al.* Górski et al. [2005][1]



(a) Equiangular face generation. (b) Cubed-sphere.

Figure 3.3: Cubed-sphere by *et. al.* Ronchi et al. [1996][2].

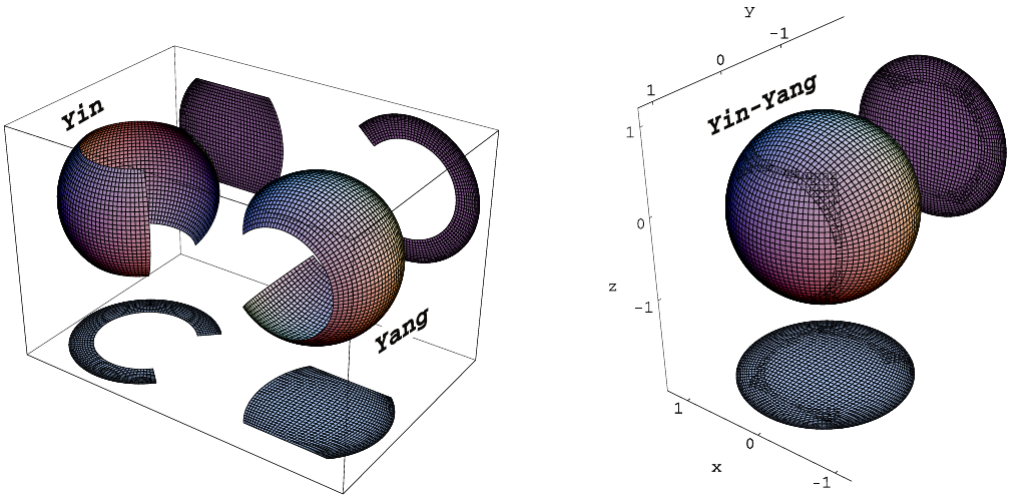
pixels.

SIFT feature point extraction is a more studied problem in omnidirectional images. Cruz-Mota et al. [2012] proposed the sSIFT algorithm to compute feature points in omnidirectional

[1]Source: http://healpix.jpl.nasa.gov/images/gorski_f1.jpg. Image copyright policy: http://www.jpl.nasa.gov/imagepolicy/

[2]Copyright 1996 by Elsevier. License for use in this thesis obtained on February 14, 2017.

[3]Copyright 2004 by John Wiley and Sons. License for use in this thesis obtained on February 13, 2017.



(a) Grid components. (b) Sphere coverage with face overlapping.

Figure 3.4: Yin-Yang grid by Kageyama and Sato [2004][3].

images. sSIFT maps the ominidirectional images to the YAWTB[4] Matlab toolbox sphere discretization to compute the scale-space using spherical harmonics. Local extrema in the scale-space are detected and feature point descriptors are computed. Hansen et al. [2010] also use spherical harmonics for computing the scale-space of the image. Filter convolutions are implemented on a stereographic projection of the camera image. SIFT descriptors are computed by projecting a neighborhood of each keypoint on a local 2D plane. The histogram of gradient descriptor is computed from this plane. The use of spherical harmonics to compute the scale-space is computationally expensive. Puig et al. [2014] define the Laplace-Beltrami operator on the sphere to approximate the difference of Gaussians operation. The scale-space can then be constructed as a series of convolutions by the Laplace-Beltrami operator applied on the original camera image.

There are also calibration toolbox to estimate camera parameters for an omnidirectional camera. Such toolbox estimate the parameters of the camera lenses and mirrors and provide a unit 3D unit vector with the bearing direction of each pixel. Figure 3.1 was created using the Calibration toolbox of Schönbein et al. [2014]. Figure 3.13 was created using the calibration toolbox of Scaramuzza et al. [2006].

3.3 Geometry

This section introduces relevant geometric definitions used across this chapter. Let $\boldsymbol{\eta} = (x, y, z) \in S^2$ be a unit vector corresponding to the embedding of spherical coordinates in $\mathbb{R}^3$. A point $\mathbf{x} \in \mathbb{R}^3$ projects to sphere space as

$$\boldsymbol{\eta} = \frac{\mathbf{x}}{\|\mathbf{x}\|} \tag{3.1}$$

[4] http://sites.uclouvain.be/ispgroup/yawtb/

The choice of embedded coordinates eliminates coordinate singularities that appear at the poles when using standard (zenith, azimuth) spherical coordinates.

The associated tangent space at $\boldsymbol{\eta}$ is

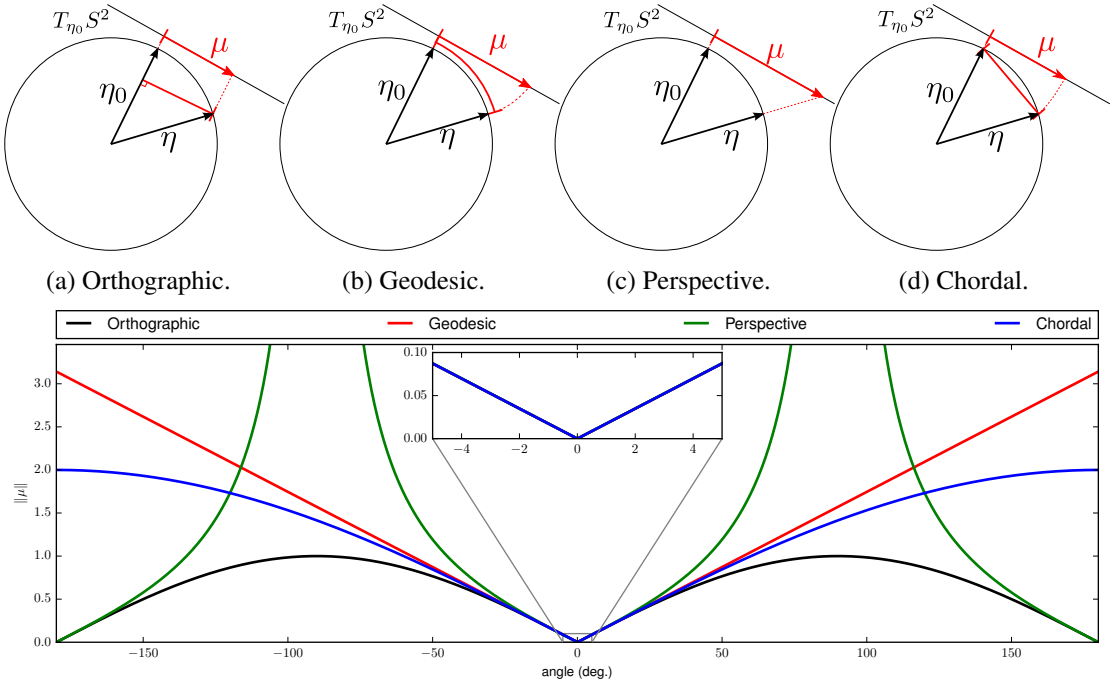
$$T_\eta S^2 = \{\boldsymbol{\mu} \in \mathbb{R}^3 \mid \langle \boldsymbol{\eta}, \boldsymbol{\mu} \rangle = 0\} \tag{3.2}$$

and is characterized by the projection matrix $\mathbf{P}_\eta : \mathbb{R}^3 \to T_\eta S^2$, computed as

$$\mathbf{P}_\eta = (\mathbf{I} - \boldsymbol{\eta}\boldsymbol{\eta}^\top) \tag{3.3}$$

In order to apply "linear" filtering to images defined on the sphere, it is natural to consider local representations of the image on a tangent space of the sphere and apply filter kernels directly in tangent coordinates. Moreover, quantities such as image gradient or optical flow are vector fields $V : S^2 \to T_\eta S^2$ defined in tangent space. To implement these operations, we need a set of functions that map points from the sphere to tangent space and *vice versa*. The term *projection-retraction class* is used to denote a pair of such functions. A retraction function is a function that maps points from the tangent space of an arbitrary manifold back to the manifold Absil et al. [2009]. For the particular case of the sphere manifold, there are at least four projection-retraction classes: *orthographic, projective, geodesic* and *chordal*.

Let $\boldsymbol{\eta}_0 \in S^2$ denote a reference point in the sphere. Denote $\mu(\boldsymbol{\eta}; \boldsymbol{\eta}_0) : S^2 \to T_{\eta_0} S^2$ to be the projection function mapping point $\boldsymbol{\eta}$ onto $T_{\eta_0} S^2$, and $\mu^{-1}(\boldsymbol{\mu}; \boldsymbol{\eta}_0) : T_{\eta_0} S^2 \to S^2$ to be the retraction function. The four projection-retraction classes are defined as:



(a) Orthographic. (b) Geodesic. (c) Perspective. (d) Chordal.

(e) Magnitude of projection functions accordint to angular separation between two points on the sphere.

Figure 3.5: Projection-retraction classes on the sphere.

1. **Orthographic:** This class takes the component of $\boldsymbol{\eta}$ orthogonal to reference point $\boldsymbol{\eta}_0$ as the projection vector onto $T_{\eta_0}S^2$. The projection function $\mu_{\text{oth}}(\boldsymbol{\eta};\boldsymbol{\eta}_0)$ is simply defined as

$$\mu_{\text{oth}}(\boldsymbol{\eta};\boldsymbol{\eta}_0) = \mathbf{P}_{\eta_0}\boldsymbol{\eta} \tag{3.4}$$

where $\mathbf{P}_{\eta_0}$ is the tangent space projection matrix computed in Equation (4.10). For convenience, denote as $\boldsymbol{\nu}$ the normalized vector indicating the direction of projection. For a given point $\boldsymbol{\eta} \neq \boldsymbol{\eta}_0$, it is computed as

$$\nu(\boldsymbol{\eta};\boldsymbol{\eta}_0) = \frac{\mu_{\text{oth}}(\boldsymbol{\eta};\boldsymbol{\eta}_0)}{\|\mu_{\text{oth}}(\boldsymbol{\eta};\boldsymbol{\eta}_0)\|} \tag{3.5}$$

The retraction function $\mu^{-1}_{\text{oth}}(\boldsymbol{\mu};\boldsymbol{\eta}_0)$ takes a point $\boldsymbol{\mu} \in T_{\eta_0}S^2$ and retracts it into the sphere as

$$\mu^{-1}_{\text{oth}}(\boldsymbol{\mu};\boldsymbol{\eta}_0) = \boldsymbol{\mu} + \boldsymbol{\eta}_0\sqrt{1-\boldsymbol{\mu}^\top\boldsymbol{\mu}} \tag{3.6}$$

2. **Geodesic:** The geodesic class takes the angular separation between $\boldsymbol{\eta}$ and $\boldsymbol{\eta}_0$ as distance metric between both points in tangent space. The geodesic projection is calculated as

$$\mu_{\text{geo}}(\boldsymbol{\eta};\boldsymbol{\eta}_0) = \arccos\left(\langle\boldsymbol{\eta},\boldsymbol{\eta}_0\rangle\right)\boldsymbol{\nu} \tag{3.7}$$

where $\boldsymbol{\nu}$ is computed using Equation (3.5).

The retraction function $\mu^{-1}_{\text{geo}}(\boldsymbol{\mu};\boldsymbol{\eta}_0)$ is calculated by rotating reference vector $\boldsymbol{\eta}_0$ by an angle $\|\boldsymbol{\mu}\|$ around rotation axis

$$\mathbf{r} = \frac{\boldsymbol{\eta}_0 \times \boldsymbol{\mu}}{\|\boldsymbol{\eta}_0 \times \boldsymbol{\mu}\|} \tag{3.8}$$

Let $\mathbf{R}$ be the rotation matrix computed from $\mathbf{r}$ and θ, the geodesic retraction is then defined as

$$\mu^{-1}_{\text{geo}}(\boldsymbol{\mu};\boldsymbol{\eta}_0) = \mathbf{R}\boldsymbol{\eta}_0 \tag{3.9}$$

3. **Perspective:** The perspective class resembles a perspective camera model in which the tangent space acts as image plane. The projection function $\mu_{\text{per}}(\boldsymbol{\eta};\boldsymbol{\eta}_0)$ extends the ray $\boldsymbol{\eta}$ until it intersects the tangent plane generated by $\boldsymbol{\eta}_0$, thus satisfying

$$\langle\alpha\boldsymbol{\eta} - \boldsymbol{\eta}_0, \boldsymbol{\eta}_0\rangle = 0 \tag{3.10}$$

where α is an unknown scale factor and $\alpha\boldsymbol{\eta} - \boldsymbol{\eta}_0 \in T_{\eta_0}S^2$ is the unknown tangent vector. From Equation (3.10), $\alpha = {}^1\!/\!_{(\boldsymbol{\eta}^\top\boldsymbol{\eta}_0)}$, and the perspective projection function is written as

$$\mu_{\text{per}}(\boldsymbol{\eta};\boldsymbol{\eta}_0) = \frac{\boldsymbol{\eta}}{\boldsymbol{\eta}^\top\boldsymbol{\eta}_0} - \boldsymbol{\eta}_0 \tag{3.11}$$

The retraction function is simply calculated as the renormalization of vector $\boldsymbol{\eta}_0 + \boldsymbol{\mu}$. That is

$$\mu^{-1}_{\text{per}}(\boldsymbol{\mu};\boldsymbol{\eta}_0) = \frac{\boldsymbol{\eta}_0 + \boldsymbol{\mu}}{\|\boldsymbol{\eta}_0 + \boldsymbol{\mu}\|} \tag{3.12}$$

4. **Chordal:** This class uses the Euclidian distance between two points in the sphere as distance metric in tangent space. The projection function is

$$\mu_{\text{cho}}(\boldsymbol{\eta};\boldsymbol{\eta}_0) = \|\boldsymbol{\eta}_0 - \boldsymbol{\eta}\|\boldsymbol{\nu} \tag{3.13}$$

where unit vector $\boldsymbol{\nu}$ is calculated using Equation (3.5).

For the retraction function, the approach is the same as for the geodesic case. A rotation matrix $\mathbf{R}$ is constructed using rotation axis from Equation (3.8) and rotation angle $\theta = 2\sin(\|\boldsymbol{\mu}\|/2)$.

$$\mu^{-1}_{\text{cho}}(\boldsymbol{\mu};\boldsymbol{\eta}_0) = \mathbf{R}\boldsymbol{\eta}_0 \tag{3.14}$$

Figure 3.5e plots the magnitude of projected points $\boldsymbol{\mu}$ as a function of angle between $\boldsymbol{\eta}_0$ and $\boldsymbol{\eta}$ for each projection class. Notice that for small angles, the difference between projection functions is negligible. The choice of a particular projection-retraction class is best considered problem dependent. Thanks to its numerical simplicity, the orthographic class is selected to map points in a small neighborhood around $\boldsymbol{\eta}_0$, and the geodesic class is chosen to deal with points located far away from the reference.

3.3.1 Orthonormal coordinate system

Considering that points $\boldsymbol{\mu} \in T_\eta S^2$ only have two degrees of freedom, it is convenient to express the 3-component vectors in terms of a 2D coordinate system. Let $\mathbf{B}_{\eta_0} \in \mathbb{R}^{2\times 3}$ denote the orthonormal basis transform that converts a vector $\boldsymbol{\mu} \in T_{\eta_0} S^2$ to 2-component vectors. The name *beta coordinates* is used to refer to the 2D representation of points in tangent space. The beta coordinate $\boldsymbol{\beta} \in \mathbb{R}^2$ of a vector $\boldsymbol{\mu} \in T_{\eta_0} S^2$ is computed as

$$\boldsymbol{\beta} := \begin{pmatrix}\beta_1 \\ \beta_2\end{pmatrix} = \mathbf{B}_{\eta_0}\boldsymbol{\mu} \tag{3.15}$$

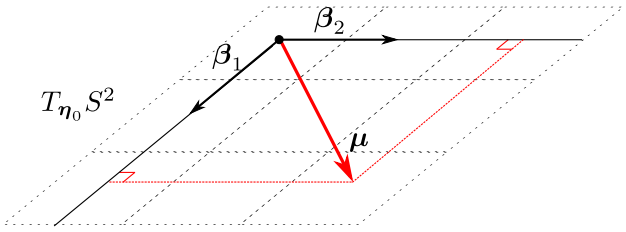


Figure 3.6: Orthonormal coordinates in tangent space $T_{\eta_0} S^2$.

Matrix $\mathbf{B}_{\eta_0}$ is constructed by taking any normalized vector $\boldsymbol{\nu}_1 \in T_{\eta_0} S^2$ using Equation (3.5) and creating a second vector $\boldsymbol{\nu}_2 = \boldsymbol{\eta}_0 \times \boldsymbol{\nu}_1$ orthogonal to both $\boldsymbol{\eta}_0$ and $\boldsymbol{\nu}_1$

$$\mathbf{B}_{\eta_0} = \begin{pmatrix}\boldsymbol{\nu}_1^\top \\ \boldsymbol{\nu}_2^\top\end{pmatrix} \tag{3.16}$$

The inverse operation, that is, transforming from $\boldsymbol{\beta}$ to its 3-vector representation $\boldsymbol{\mu}$ is

achieved by

$$\boldsymbol{\mu} = \mathbf{B}_{\eta_0}^{\top}\boldsymbol{\beta} \tag{3.17}$$

In practice, the beta coordinate representation creates a coordinate system in which it is possible to define image operators in terms of convolutions in 2 axis. Since the coordinate system is orthogonal, properties such as separability can be used to speed up computations.

3.4 The Spherepix Data Structure

The objective of the Spherepix data structure is to design a discretization of the sphere in which residual error in orthogonality and equal spacing properties of the pixelation are minimized and distributed across the data structure as evenly as possible. If these two properties are satisfied, then the accuracy of low-level shift invariant image processing operations on spherepix will not suffer significantly, opening the door to high-speed image processing of omnidirectional images.

The Spherepix data structure is a mosaic of K two-dimensional memory grids storing 3D coordinates of points lying on the sphere. Each grid is denoted by E_k for $k = 1, \ldots, K$, and sphere coordinates $\boldsymbol{\eta}_{ij}$ within each patch are indexed by (row, column) integers (i, j).

A coordinate regularization procedure is applied to each patch to equalize the location of each pixel. The initial condition is the equiangular cubic discretization of the sphere of Ronchi et al. [1996] with resolution $N \times N$, where N is a user defined parameter. The regularization consists in the simulation of a mass-spring-damper system. Each pixel $\boldsymbol{\eta}_{ij}$ in a patch is assigned an equal mass M and is connected to its 8-neighborhood in tangent space through springs with elastic constant K, as illustrated in Figure 3.7. The springs' rest elongation L is constant throughout the simulation and is equal to the distance between two points at the center of the initial grid

$$L = \|\mu_{\text{oth}}(\boldsymbol{\eta}_{N/2+1,N/2}; \boldsymbol{\eta}_{N/2,N/2})\| \tag{3.18}$$

In the 8-neighborhood of each point, the rest elongation is set to L to points in the column or row axis, and $\sqrt{2}L$ to points on the diagonal axis. This choice of elongations enforces the spherical pixels to form a square lattice in tangent space. Finally, a velocity damper is added to each point to dissipate energy over time in order to reach a stable state.

The idea behind this system is that the steady state solution of the dynamic system will create a lattice of points that approximately holds the properties of orthogonality and equal spacing. This construction differs to that of Putman and Lin [2007] in that they used linear and torsion springs to impose equidistant and orthogonality constrains to the system. Moreover, Putman and Lin forced the border pixels of their dynamic system to be fixed in space, thus not allowing the size of the patches to change, affecting the quality of the regularized grids.

The resulting Euler-Lagrange system creates a system of ordinary differential equations that can be solved numerically using Euler integration method. Appendix B provides the details of this system and its numerical solution. The simulation runs independently for each patch for a fixed number of iterations or until the velocity of each point mass is below a certain threshold.

To improve the regularity of the grids, in particular the distance between neighbors, it is possible to apply the regularization procedure to subdivisions of the initial grid. Overlapping

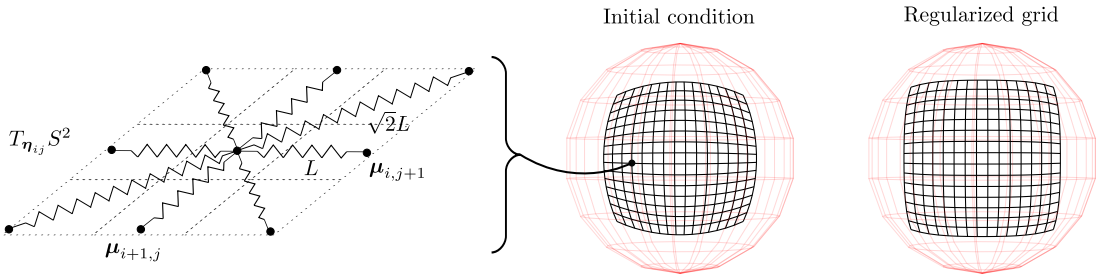
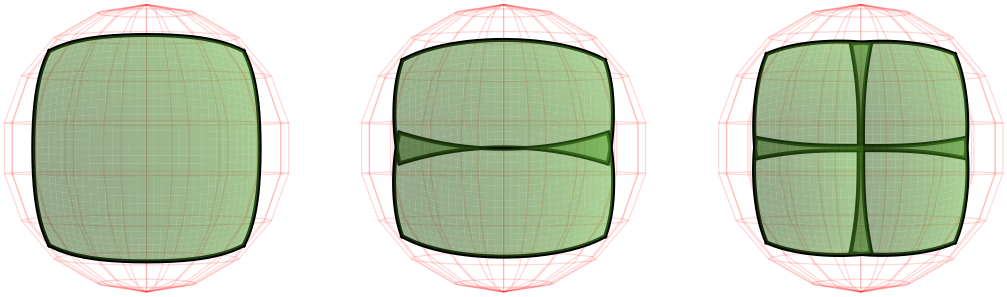


Figure 3.7: Mass-spring system elements for sphere point $\boldsymbol{\eta}_{ij}$. Springs connect the 8-neighborhood of each pixel $\boldsymbol{\eta}_{ij}$ in the tangent space $T_{\eta_{ij}} S^2$. This spring configuration constrains neighboring points to be equidistant and orthogonal in the tangent space. The dynamic system is run to create a regularized grid that approximately satisfy these constrains (Figure 3.9).

between patches is ensured by slightly increasing the point separation length at each subdivision mode. Figure 3.8 illustrates two subdivisions of a single face resulting in four overlapping grid patches, and Figure 3.9 shows the angle between neighboring points and its relative distance variation before and after the regularization for a patch of size 1024×1024. The regularization procedure is time consuming, taking several minutes on CPU to complete for resolutions as big as 1024×1024. The regularized coordinate grids can be stored in disk and be loaded for later use.



(a) mode 1 (b) mode 2 (c) mode 3

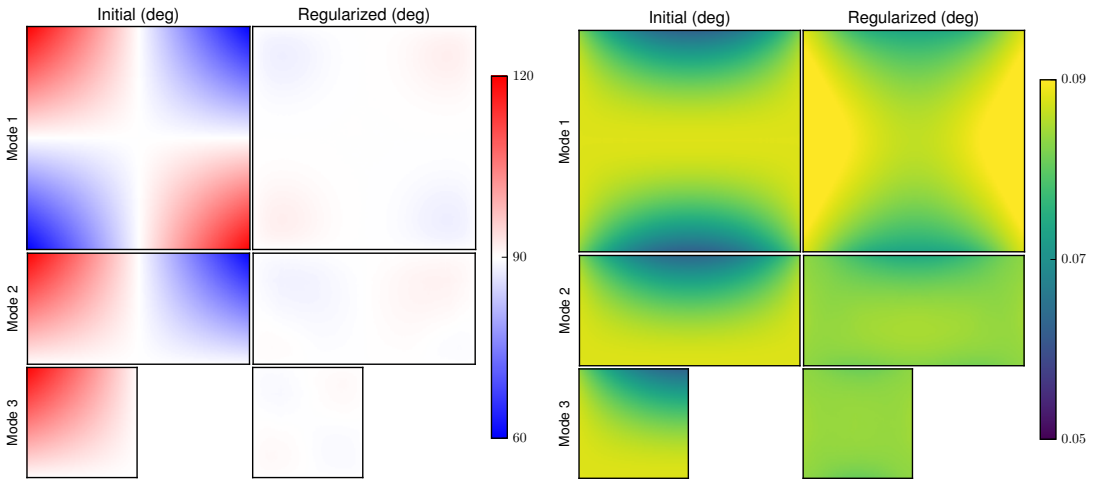
Figure 3.8: Patch subdivision modes. Subdivisions of the initial patch improves the regularity of the grid. The shaded areas in the figures indicate overlapping of the subdivision patches.

3.4.1 Pixelation properties

The following properties extracted from the regularized pixelation patches are relevant. First, $\Delta\mu_{ij}$ denotes the pixel separation. It is calculated as

$$\Delta\mu_{ij} = \|\mu_{\text{oth}}(\boldsymbol{\eta}_{i,j+1}; \boldsymbol{\eta}_{ij})\| \tag{3.19}$$

Next, let $\mathbf{B}_{ij}$ be the orthonormal basis matrix that maps vectors $\boldsymbol{\mu} \in T_{\eta_{ij}} S^2$ to 2D grid coordinates. Row elements of $\mathbf{B}_{ij}$ correspond to the normalized projection of neighboring column



(a) Angle $\theta = \arccos\left(\frac{\langle \mu_{\text{oth}}(\boldsymbol{\eta}_{i,j+1};\boldsymbol{\eta}_{ij}), \mu_{\text{oth}}(\boldsymbol{\eta}_{i+1,j};\boldsymbol{\eta}_{ij})\rangle}{\|\mu_{\text{oth}}(\boldsymbol{\eta}_{i,j+1};\boldsymbol{\eta}_{ij})\| \|\mu_{\text{oth}}(\boldsymbol{\eta}_{i+1,j};\boldsymbol{\eta}_{ij})\|}\right)$ between orthogonal neighbors expressed in degrees. (b) Distance $\|\mu_{\text{oth}}(\boldsymbol{\eta}_{i,j+1};\boldsymbol{\eta}_{ij})\|$ between neighbors expressed in degrees.

Figure 3.9: Angle and distance between neighbors before and after coordinate regularization for different subdivision modes.

and row pixels onto $T_{\eta_{ij}}S^2$. That is

$$\mathbf{B}_{ij} = \begin{pmatrix} \mu_{\text{oth}}(\boldsymbol{\eta}_{i,j+1};\boldsymbol{\eta}_{ij})^\top / \|\mu_{\text{oth}}(\boldsymbol{\eta}_{i,j+1};\boldsymbol{\eta}_{ij})\| \\ \mu_{\text{oth}}(\boldsymbol{\eta}_{i+1,j};\boldsymbol{\eta}_{ij})^\top / \|\mu_{\text{oth}}(\boldsymbol{\eta}_{i+1,j};\boldsymbol{\eta}_{ij})\| \end{pmatrix} \tag{3.20}$$

This choice of $\mathbf{B}_{ij}$ creates a direct relationship between 2D memory coordinates and tangent space in the sphere, which is fundamental for fast interpretation of image operations on the sphere. Moreover, thanks to the spring regularization method applied the Spherepix patches, the row elements of $\mathbf{B}_{ij}$ are approximately orthogonal. Consequently, it is possible to interpret the output of convolutions in row and column axes as quantities in tangent space.

The transformation of tangent space vectors to 2D grid coordinates $\boldsymbol{\beta} \in \mathbb{R}^2$ follows

$$\boldsymbol{\beta} = \frac{1}{\Delta\mu_{ij}} \mathbf{B}_{\boldsymbol{\eta}_{ij}} \boldsymbol{\mu} \tag{3.21}$$

while the reverse mapping, that is, from grid to tangent coordinates, is

$$\boldsymbol{\mu} = \Delta\mu_{ij} \mathbf{B}_{\boldsymbol{\eta}_{ij}}^\top \boldsymbol{\beta} \tag{3.22}$$

3.4.2 Coordinate interpolation

The regularization procedure applied to each pixelation patch increases its surface area. As a result, there are regions of the sphere covered by more than one patch, as illustrated in Figure 3.8. In general, image processing algorithms are applied to each patch separately (Section 3.5). To avoid artificial border artifacts created by algorithms requiring pixels outside image boundaries, each patch is surrounded with a belt of pixel interpolation coordinates. Each point

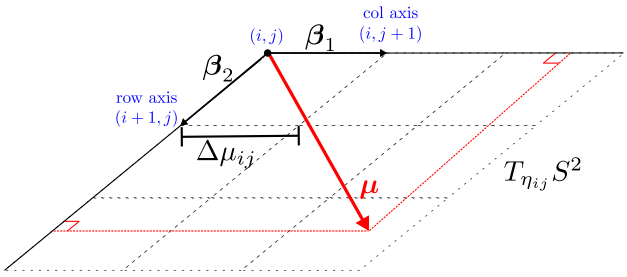


Figure 3.10: Orthonormal grid coordinates in tangent space $T_{\eta_{ij}} S^2$.

in this belt stores spherical coordinates used to interpolate image values from neighboring patches. Tangent vectors are computed using Equation (3.22) with grid coordinates beyond patch boundaries. The corresponding spherical coordinates are recovered by retracting these tangent vectors into the sphere.

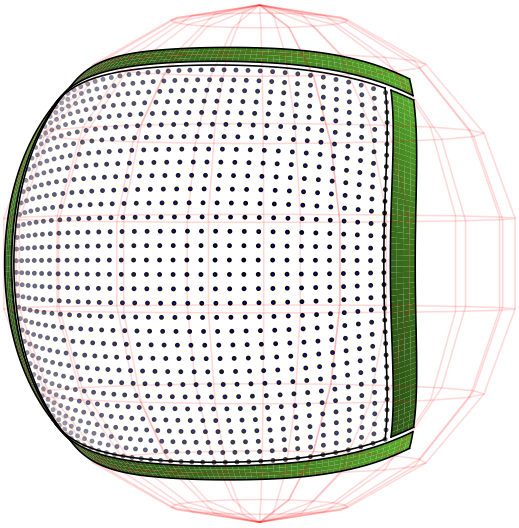


Figure 3.11: Patch interpolation belt. Image values for pixels outside patch boundaries are obtained by interpolation into neighboring patches.

The spherical coordinates in the interpolation belt are the input to a search algorithm to find pixel coordinates in neighboring patches. The algorithms follows a quad-tree search strategy to find the closest point in the grid to the input query point. At each step, the grid is recursively subdivided in four quadrants and the closest quadrant to the query point is selected each time, until the search area reaches a size of one pixel. The pseudo-code of this procedure is described in Algorithm 1. Notice that the geodesic projection-retraction class is used in the search loop. As mentioned in Section 3.3, this class is useful to perform operations when points are far away from the origin.

Algorithm 1 Interpolation coordinate search for point $\boldsymbol{\eta}$ inside patch E.

input: E, height, width, $\boldsymbol{\eta}$
$p \leftarrow (\lfloor \text{height}/2 \rfloor, \lfloor \text{width}/2 \rfloor)$ ▷ initial candidate.
$q \leftarrow (\lfloor \text{height}/4 \rfloor, \lfloor \text{width}/4 \rfloor)$ ▷ subdivision factor.
while $q_i > 0$ **or** $q_j > 0$ **do**
  $u \leftarrow \infty$
  $p_{\text{new}} \leftarrow (0, 0)$
  for $r \in \{-q_i, q_i\}$ **do**
    for $c \in \{-q_j, q_j\}$ **do**
      $\boldsymbol{\eta}_{rc} \leftarrow E[p_i + r, p_j + c]$
      $\boldsymbol{\mu}_{rc} \leftarrow \mu_{\text{geo}}(\boldsymbol{\eta}; \boldsymbol{\eta}_{rc})$ ▷ geodesic projection
      if $\|\boldsymbol{\mu}_{rc}\| \leq u$ **then**
        $u \leftarrow \|\boldsymbol{\mu}_{rc}\|$
        $p_{\text{new}} \leftarrow p + (r, c)$
      end if
    end for
  end for
  $p \leftarrow p_{\text{new}}$
  $q \leftarrow (\lfloor q_i/2 \rfloor, \lfloor q_j/2 \rfloor)$
end while
$\boldsymbol{\eta}_p \leftarrow E[p_i, p_j]$
$\mathbf{B}_{ij}, \Delta\boldsymbol{\mu}_{ij} \leftarrow$ basis at p ▷ Eq. (3.20), (3.19).
$\boldsymbol{\mu} \leftarrow \mu_{\text{geo}}(\boldsymbol{\eta}; \boldsymbol{\eta}_p)$
$\hat{p} \leftarrow p + \frac{1}{\Delta\boldsymbol{\mu}_p} \mathbf{B}_{\boldsymbol{\eta}_p} \boldsymbol{\mu}$ ▷ Eq. (3.21).
output: $\hat{p}$

3.5 Low-level Image Processing Operations

Image processing algorithms in the Spherepix patches comprise three distinct components. First, camera images are mapped onto the Spherepix patches using the calibrated camera model of the capturing system. Secondly, image processing subroutines run on the separate grid patches using interpolated data near the boundaries. Finally, after each image subroutine, the data from overlapping grid patches is reconciled to produce a homogeneous representation of the output.

3.5.1 Camera mapping

To map images onto spherepix patches, calibrated camera models are used to compute spherical coordinates for each pixel in the captured image. This grid is the input argument E to Algorithm 1, which runs for each pixel $\boldsymbol{\eta}_{ij}$ of each regularized Spherepix patch to find its corresponding interpolation coordinate in the camera image grid. The interpolation coordinates are computed offline and stored for use in real-time pipelines. In practice, the interpolation algorithm can take advantage of the texture units present in current GPU hardware to speed up computations.

Figure 3.12 shows the spherical coordinates for a perspective, wide-angle, fish-eye and

catadioptric camera models and their distance and angle between neighboring pixels. For the later three models, Scaramuza's calibration toolbox Scaramuzza et al. [2006] is used to find the spherical coordinates. The omnidirectional camera model shows the highest variation in pixel density compared to the other three models. The central region of the image surface is more coarsely sampled than the exterior parts. Regarding angle between neighbors, the fish-eye lens camera presents the largest angular distortion among all camera models.

Figure 3.13 illustrates the mapping process for one of the test images in the calibration toolbox of Scaramuzza *et. al.* [5]. The original image (Figure 3.13a) has resolution 1024×768 pixels and minimum and maximum angular separation of 0.05 and 0.43 degrees respectively (Figure 3.12d). Spherepix patches are constructed with resolution 512×512 pixels and angular separation of 0.15 deg. This is a compromise between patch resolution and similarity to the original angular separation. The GPU image interpolation routine takes on average 0.03 ms to compute the brightness value for each patch (Nvidia GTX 780 card).

Due to the uneven pixel distribution in the original image, there are some interpolation artifacts appearing on the mapped image patches, in particular, the apparent blurring effect in the bottom patch in Figure 3.13b. These artifacts could be solved, for example, using a multi-camera system where each camera records a smaller region of the sphere. In that scenario, the Spherepix data structure can be used to unify all these images in a single representation in which image processing algorithms can be applied.

3.5.2 Low-level image processing routines

Gaussian filtering and image gradient computation are used as examples of operations working with scalar and vectors fields on Spherepix images, although it should be stressed that the Spherepix data structure is suitable for many other low-level image processing routines.

3.5.2.1 Gaussian filtering

Let $G(\boldsymbol{\mu}, \sigma)$ denote the zero-mean Gaussian function with standard deviation σ applied to vector $\boldsymbol{\mu} \in T_\eta S^2$

$$G(\boldsymbol{\mu}, \sigma) = \alpha e^{-\frac{\langle \boldsymbol{\mu}, \boldsymbol{\mu} \rangle}{2\sigma^2}} \tag{3.23}$$

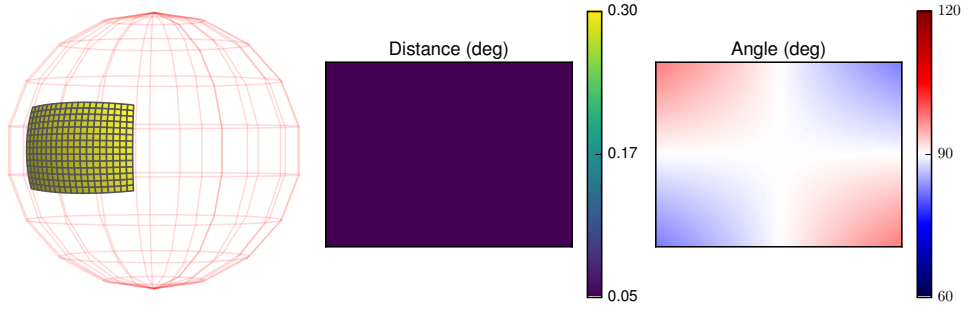
with $\alpha = {}^1\!/_{2\pi\sigma^2}$. To obtain an efficient implementation of the Gaussian operator by means of separable convolutions, it is possible to approximate Equation (3.23) in terms of its beta-coordinates equivalent and split the function in terms of 2 Gaussian functions for $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ as

$$G(\boldsymbol{\beta}, \sigma) = \alpha e^{-\frac{\langle \boldsymbol{\beta}, \boldsymbol{\beta} \rangle}{2\sigma^2}} \tag{3.24}$$

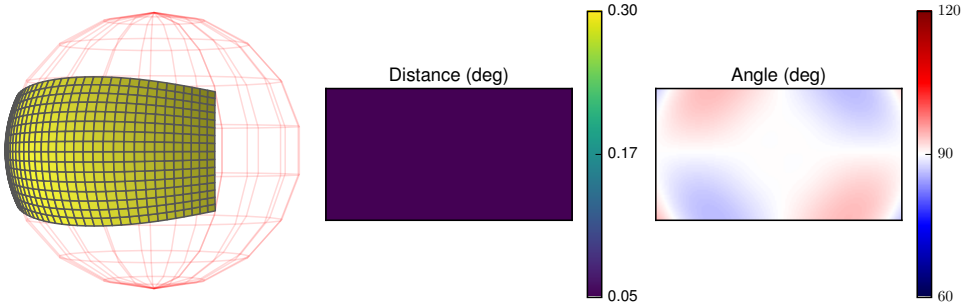
$$\approx \sqrt{\alpha} e^{-\frac{\boldsymbol{\beta}_1^2}{2\sigma^2}} \sqrt{\alpha} e^{-\frac{\boldsymbol{\beta}_2^2}{2\sigma^2}} \tag{3.25}$$

Since $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ match the column and row axes of each Spherepix patch in memory, the implementation of the Gaussian filter is equal to that of planar images.

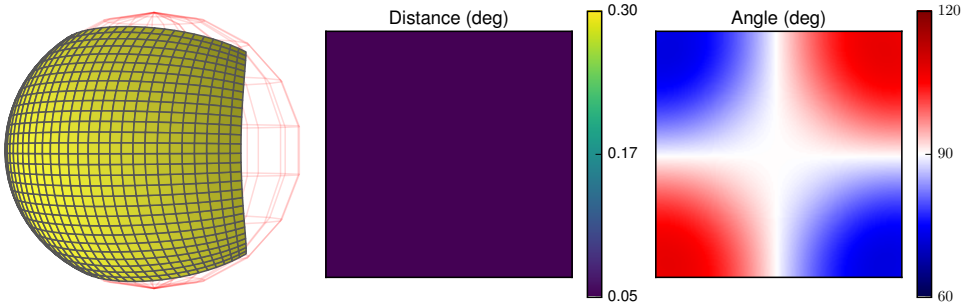
[5] https://sites.google.com/site/scarabotix/ocamcalib-toolbox



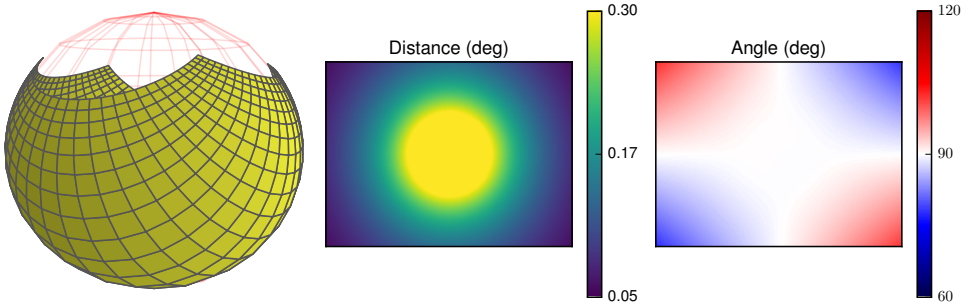
(a) Perspective



(b) Wide angle

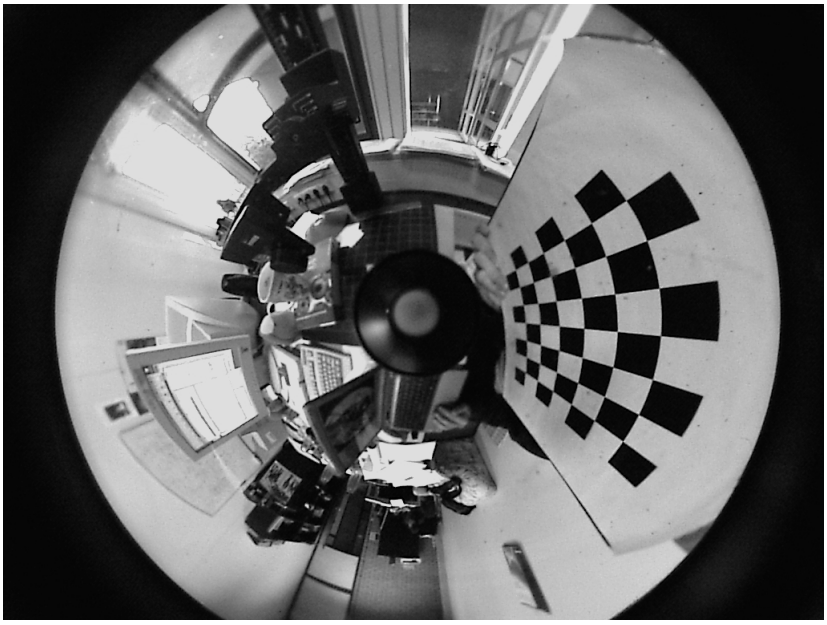


(c) Fish eye

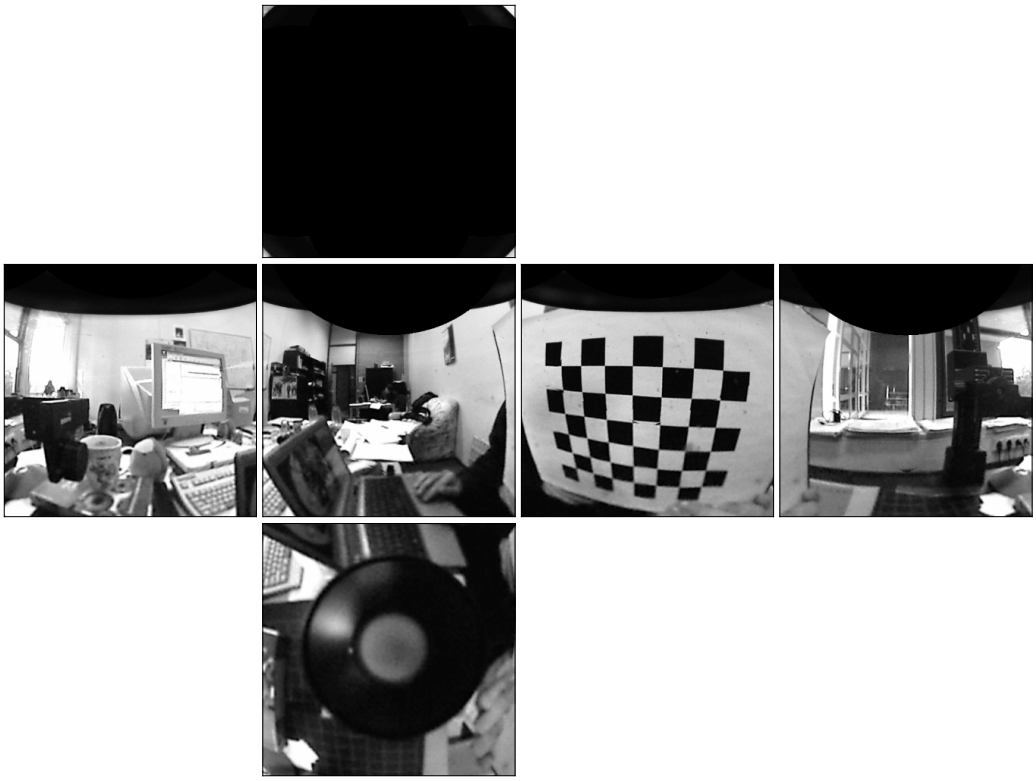


(d) Omnidirectional

Figure 3.12: Mapping of different camera models to the sphere plus pixel separation and angle between neighbors.



(a) Input image.



(b) Spherepix image patches.

Figure 3.13: Image mapping from omnidirectional camera (Scaramuzza *et. al.* Scaramuzza et al. [2006]) to Spherepix patches.

3.5.2.2 Image gradient

Image gradient is the vector field $\frac{\partial Y}{\partial \boldsymbol{\eta}} : S^2 \rightarrow T_\eta S^2$ describing the change of image intensity Y across space. Computation is performed in two steps.

First, the gradient $\frac{\partial Y}{\partial \boldsymbol{\beta}}$ expressed in beta-coordinates is computed by convolving Y with the Sobel filter masks

$$S_{\boldsymbol{\beta}_1} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{3.26}$$

$$S_{\boldsymbol{\beta}_2} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.27}$$

obtaining two-dimensional gradient vector field

$$\frac{\partial Y}{\partial \boldsymbol{\beta}} := \begin{pmatrix} \frac{\partial Y}{\partial \boldsymbol{\beta}_1} \\ \frac{\partial Y}{\partial \boldsymbol{\beta}_2} \end{pmatrix} = \begin{pmatrix} Y * S_{\boldsymbol{\beta}_1} \\ Y * S_{\boldsymbol{\beta}_2} \end{pmatrix} \tag{3.28}$$

Second, $\frac{\partial Y}{\partial \boldsymbol{\beta}}$ is transformed to its equivalent tangent vector representation following Equation (3.22), obtaining

$$\frac{\partial Y_{ij}}{\partial \boldsymbol{\eta}} = \Delta\mu_{ij} \mathbf{B}_{\boldsymbol{\eta}_{ij}}^\top \frac{\partial Y_{ij}}{\partial \boldsymbol{\beta}} \tag{3.29}$$

Figure 3.14 illustrates the two processing steps. Notice that the magnitude of the 3D gradient vector is affected by pixel separation $\Delta\mu_{ij}$. This change in scale needs to be considered in algorithms manipulating vector and scalar fields together.

3.5.3 Patch reconciliation

After each image processing subroutine, the output in the overlapping areas needs to be reconciled. This enforces that each Spherepix patch has the same information content about the overlapped area. The patch reconciliation process changes according to the type of image data.

For scalar fields, the reconciliation consists in averaging the output of each pixel in the overlapping regions with the interpolated output in its neighbor patch. Currently, all pixels in the overlapping region are weighted equally, although more sophisticated schemes considering the distance to patch border are possible. For vector fields, the reconciliation method applies the averaging to each 3D vector component. Thus, vector information needs to be transformed from beta to 3D tangent space coordinates before the averaging using Equation (3.22).

The same procedure can be used to render virtual views of the spherical data. In this case, the spherical coordinates of the virtual view are used to interpolate data from each Spherepix patch using Algorithm 1. The averaged output at each virtual view pixel is the average of interpolated values from all patches that overlap at that point. Figure 3.15 shows the output of creating a panoramic view from the image patches in Figure 3.13. The average masks indicates the number of patches that overlap at each virtual view pixel.

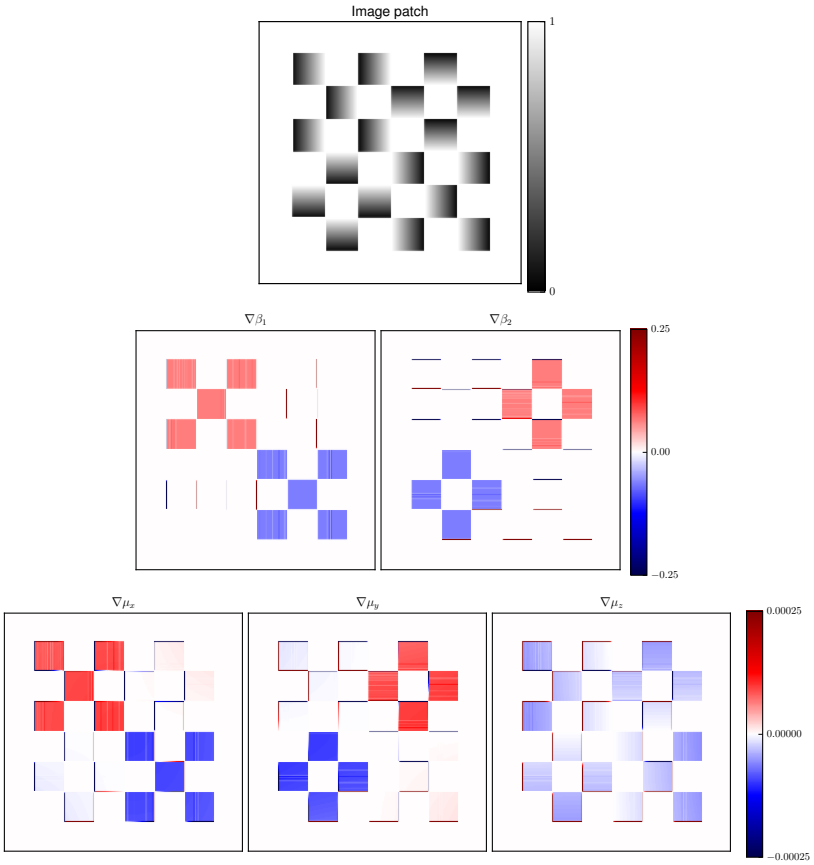


Figure 3.14: Image gradient computation. Top: wide angle image mapped to spherepix patch. Middle: gradient in beta coordinates (pixel units). Bottom: gradient in 3D tangent space coordinates.

3.6 Applications

The following are two example applications on how to use Spherepix images for standard computer vision tasks.

3.6.1 SIFT feature point extraction

The Scale Invariant Feature Transform Lowe [2004] extracts feature points from an image by finding local extrema (keypoints) in the scale-space and computes a histogram of gradients descriptor around the detected keypoints.

Applying Gaussian filters and image gradient as discussed in Section 3.5, it is straight forward to build the image scale space and compute feature point descriptors in the Spherepix formulation. In particular, the histogram of gradients can be expressed in local beta coordinates. The resulting feature descriptors are referred as *spixSIFT* features.

To validate this approach, photorealistic synthetic images rendered with Blender[6] using the

[6]Modified Barcelona pabellon from http://www.emirage.org/

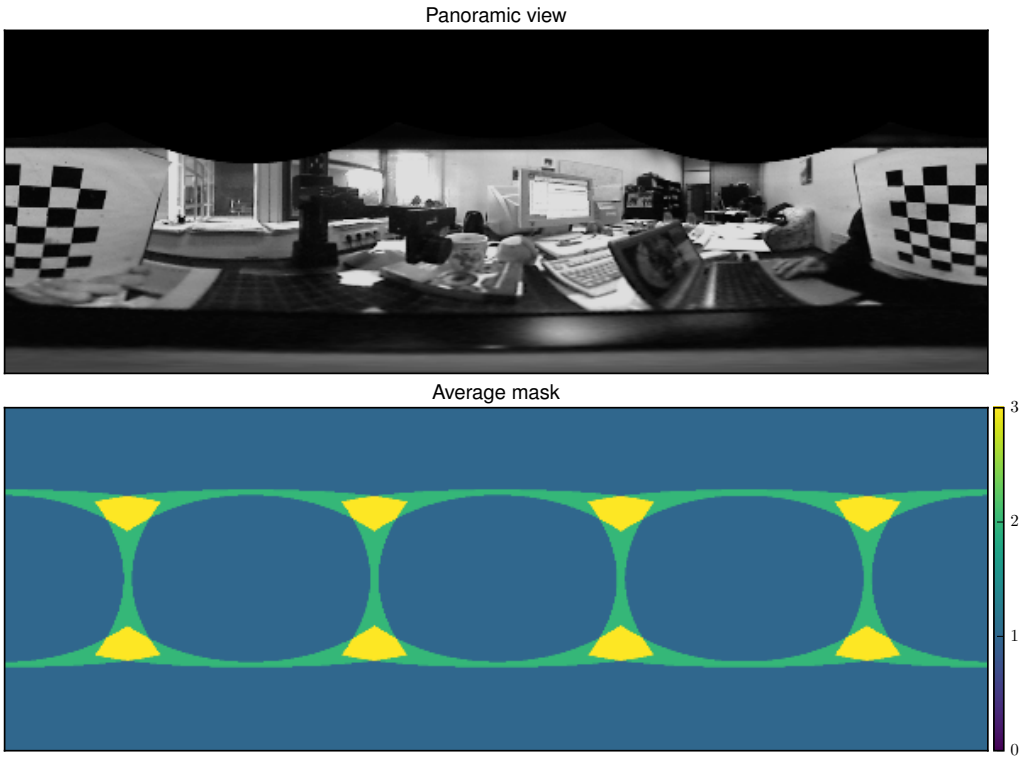


Figure 3.15: Panoramic rendering of Spherepix image.

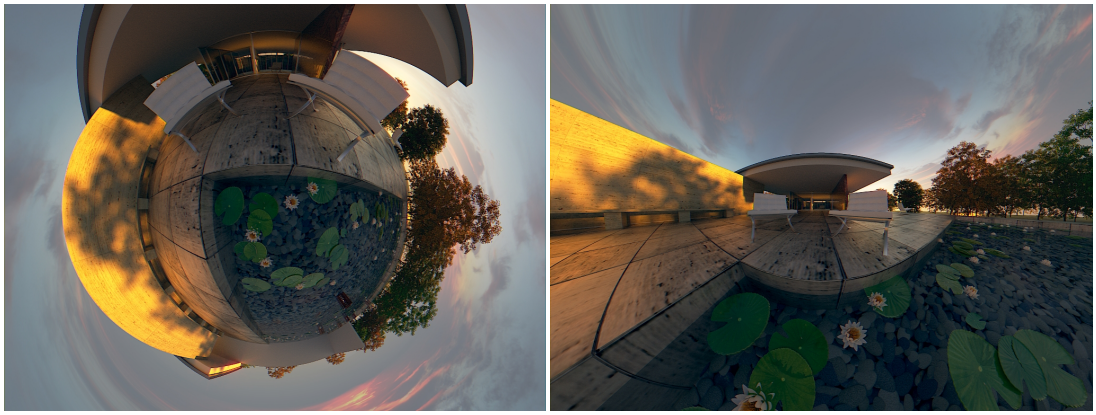
omnidirectional camera plugin of Zhang *et. al.* Zhang et al. [2016] are used. The position of the camera is fixed and the orientation in the y axis is changed from 0 to 90 degrees with step of 10 degrees. Figure 3.16 shows the initial and final rendered images.

OpenCV algorithms are used for SIFT point computation and matching on 256×256 Spherepix image patches. Brightness data is mapped using the omnidirectional camera model of Scaramuzza et al. [2006]. Duplicated keypoints in the overlapping regions of the Spherepix patches are detected and only copy is kept. Figure 3.18 shows the feature points computed on each patch as well as the mapping into the original camera image.

The following algorithms are used for comparison: SIFT applied on the original rendered image, sSIFT Cruz-Mota et al. [2012] and the proposed spixSIFT. The feature points computed by each algorithm on the first image (0 degrees rotation) are kept as ground truth. Feature points of the rotated images are matched against the reference using a descriptor distance ratio threshold of 0.7. Points that pass this matching criteria are tested to be geometrically correct. The geometric matching criteria is

$$\arccos(\langle R\boldsymbol{\eta}_{\text{gt}}, \hat{\boldsymbol{\eta}} \rangle) < 1.0\,\text{deg}$$

where $\boldsymbol{\eta}_{\text{gt}}$ is the reference point at 0 degrees rotated by matrix R to express it in the rotated image. Point $\hat{\boldsymbol{\eta}}$ is the computed keypoint from the rotated image. The value 1.0 degrees corresponds approximately to twice the maximum angular separation between pixels in the rendered image.



(a) Reference (0 degrees rotation). (b) 90 degrees rotation.

Figure 3.16: Synthetic ommnidirectional images.

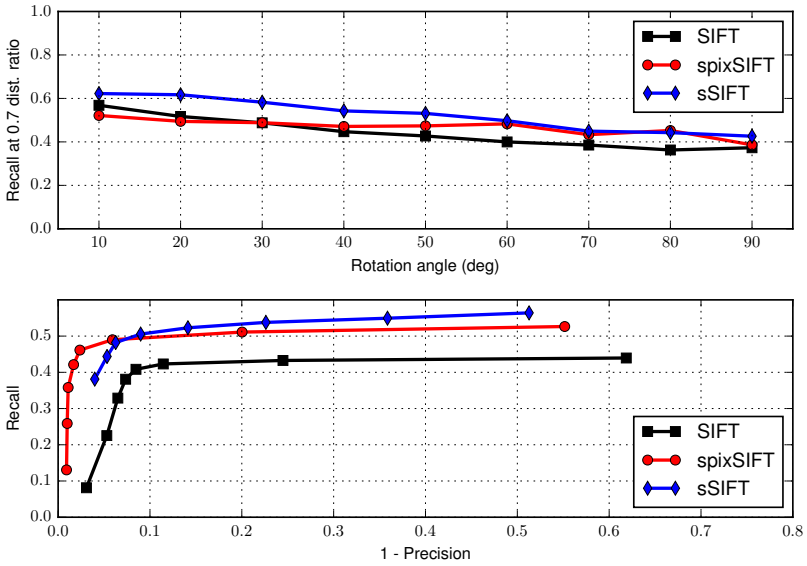


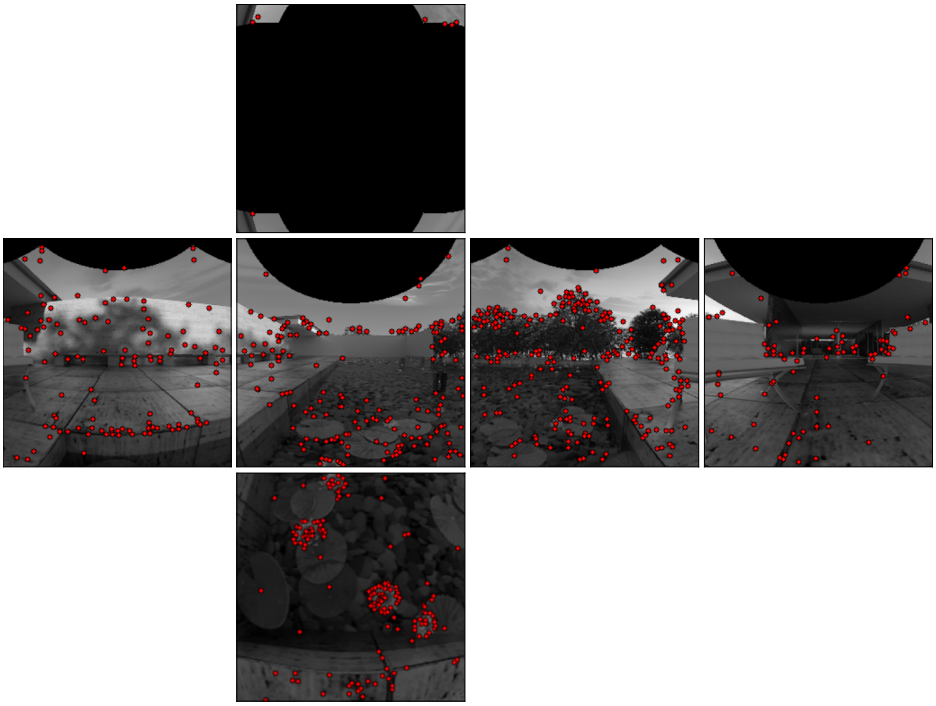
Figure 3.17: Recall vs rotation angle and Recall vs. 1 - Precision curves.

Following previous literature Hansen et al. [2010]; Cruz-Mota et al. [2012]; Puig et al. [2014], recall and one minus precision are used as performance metrics for the comparison

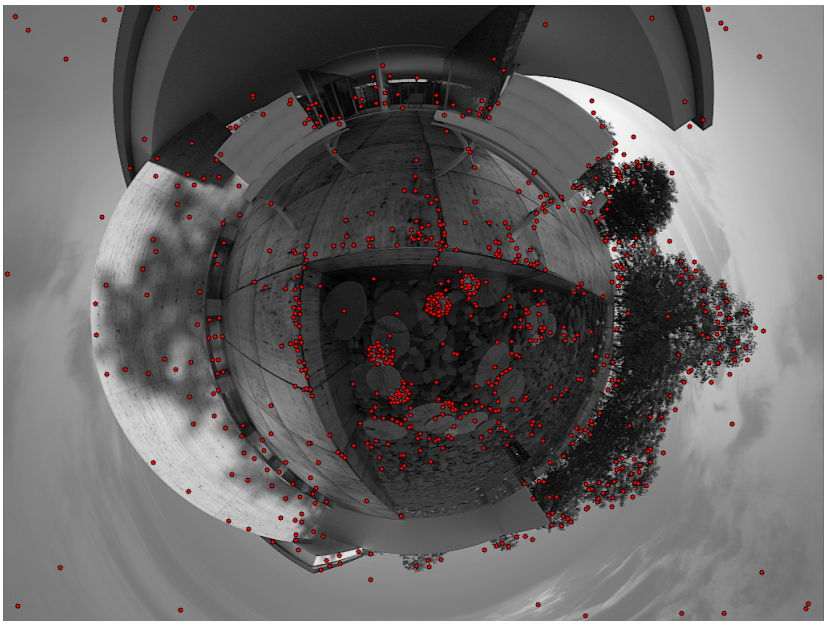
$$\text{recall} = \frac{\#\text{ total correct geometric matches}}{\text{total ground truth points}}$$

$$1 - \text{precision} = \frac{\#\text{ total false matches}}{\text{total matches}}$$

Figure 3.17 shows recall–versus–rotation angle for the descriptor distance ratio fixed at 0.7, and recall–versus–(1 - precision) for all image matches varying the distance ratio from 0.1 to 1.0. In the first figure, all algorithms do reasonably well in maintaining good recall across all angles. The proposed spixSIFT for small angles is directly comparable to classical SIFT,



(a) Feature points compute on each Spherepix patch.



(b) Feature points mapped into original camera image.

Figure 3.18: spixSIFT feature point extraction.

as expected, since both spixSIFT and SIFT run the same OpenCV algorithm. However, the geometric corrections that are integral in the Spherepix- parameterization mean that the recall performance is maintained across all angles.

In the recall–versus–(1 - precision) curve we observe the classical SIFT algorithm has both a lower recall level for moderate precision (less than 90% precision, that is, greater than 10% false matches) as well as a poor degradation in recall as precision is increased and false matches are eliminated. It is interesting to note that the spixSIFT and sSIFT algorithms have comparable recall at moderate precision and spixSIFT has clearly superior recall for very high precision, when the matching threshold is increased to ensure false matches are less than 10%. It is believed that this is due to the fact that the Spherepix representation ensures minimal distortion of the SIFT feature descriptors across the full image.

In terms of runtime performance, SIFT on the original image (1024×768) takes 240 ms on CPU. For spixSIFT, the mapping from the omnidirectional image to the six 256×256 Spherepix patches takes in total 0.085 ms on GPU plus 120 ms on CPU to extract the feature points. The decrease in total computation time for spixSIFT is due to a reduction in the total number of pixels required after mapping to Spherepix patches. This is because the full image effectively over-samples in the peripheral zone of the image due to the catadioptric distortion (see Figure 3.12d). For sSIFT, the reported runtime is 2.439×10^4 ms.

3.6.2 Dense optical flow computation

The second application example is the computation of dense optical flow on spherical images. It is possible to derive the brightness conservation equation for the case of spherical images as it was done in Chapter 2 for planar images. The equations in this section illustrate how to use standard optical flow algorithms on Spherepix images and how to interpret the output of such algorithms as vectors fields on the sphere. A complete development of these equations is provided in the Structure-flow Chapter 4.

Let $Y(\boldsymbol{\eta}, t)$ be the brightness of a spherical image at point $\boldsymbol{\eta}$ and time t. The brightness conservation assumption states that the total change in image brightness is zero, that is

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = 0 \tag{3.30}$$

Expressing Equation (3.30) in terms of its partial derivatives, one obtains PDE (3.31) for image brightness conservation on the sphere.

$$\frac{\partial Y}{\partial \boldsymbol{\eta}} \frac{\partial \boldsymbol{\eta}}{\partial t} + \frac{\partial Y}{\partial t} = 0 \tag{3.31}$$

In this case, $\frac{\partial Y}{\partial \boldsymbol{\eta}} \in T_\eta S^2$ denotes image gradient at $\boldsymbol{\eta}$, and $\frac{\partial \boldsymbol{\eta}}{\partial t} \in T_\eta S^2$ is the optical flow at $\boldsymbol{\eta}$.

Since variables in Equation (3.31) are either scalar or vector fields in tangent space, it is possible to transform Equation (3.31) to orthonormal coordinates, and express the conservation equation in terms of grid coordinates as

$$\frac{\partial Y}{\partial \boldsymbol{\beta}} \frac{\partial \boldsymbol{\beta}}{\partial t} + \frac{\partial Y}{\partial t} = 0 \tag{3.32}$$

Equation (3.32) has a similar 2D formulation as the original brightness conservation equation derived for perspective images Barron [1994]. As a result, it is possible to use any algorithm available for perspective images and compute dense optical flow on Spherepix images.

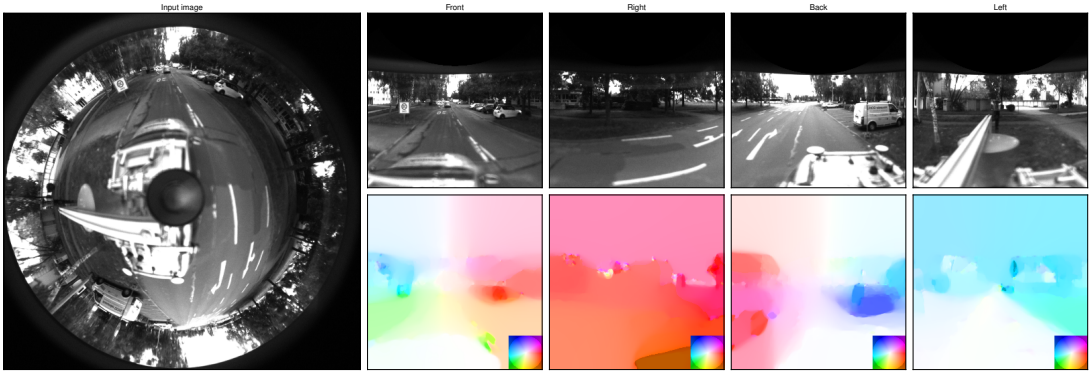


Figure 3.19: Brox et al. [2004] dense optical flow computation on Spherepix patches.

The resulting optical flow can be interpreted as a 3D tangent vector field following Equation (3.22). That is

$$\frac{\partial \boldsymbol{\eta}}{\partial t} = \Delta\mu \mathbf{B}_{\eta}^{\top} \frac{\partial \boldsymbol{\beta}}{\partial t} \tag{3.33}$$

To illustrate this, the omnidirectional image dataset of Schönbein et al. [2014] is used as input to an optical flow algorithm. This dataset provides 1400×1400 images recorded at 10 Hz. Such framerate is not sufficient for optical flow filter algorithm in Chapter 2 to give sensible results. Instead, OpenCV implementation of Brox et al. [2004] total variational method is used to demonstrate the approach.

Figure 3.19 shows the estimated optical flow in beta coordinates for image patches in the equatorial belt of the sphere. In this scene, the vehicle moves straight along the road, inducing a divergent optical flow field with the focus of expansion located in the front side of the car. Since the beta flow describes the motion in the column and row axis, it is possible to use the color wheel encoding on each patch to plot the flow field.

Figure 3.20 shows a panoramic view of the optical flow components expressed in tangent space. In particular, notice the divergence pattern in the X flow component starting in the middle of the image and vanishing at both sides of the vehicle. From there, the X-flow changes sign and converges on the rear side of the car. The Y-flow shows a regular pattern with the flow value positive almost everywhere, and can be used to infer the direction of motion.

As Brox algorithm was applied individually to each Spherepix patch using OpenCV implementation, it was not possible to use the interpolation belt of the patch to access image content from neighboring images. Moreover, the algorithm could not reconcile information from overlapping patches, as this needs to be implemented internally in Brox code. Therefore, artificial border artifacts can be visualized in Figure 3.20. In order to remove these borders, the border handling routines in OpenCV would need to be modified to access data from the Spherepix images and the reconciliation of optical flow would need to be called after each iteration of the algorithm. These implementation details will be considered in a future version of the Spherepix code.

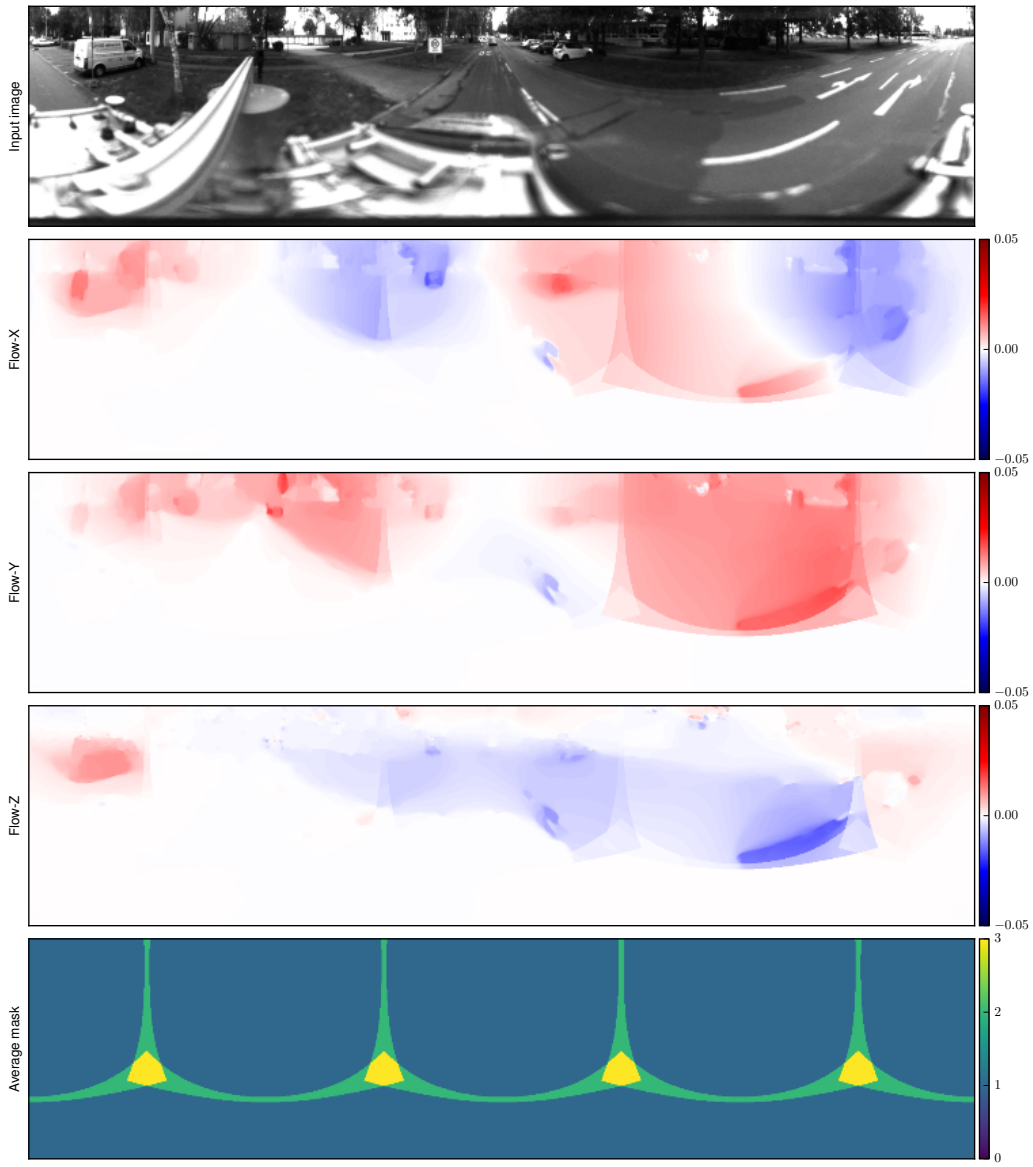


Figure 3.20: Panoramic view of optical flow in tangent space coordinates.

3.7 Summary

This chapter described the Spherepix data structure for efficient processing of spherical images. The data structure consists of a mosaic of grid patches covering the sphere's surface. Within each patch, a regularization procedure is applied to the Spherepix coordinates so that they approximate an orthogonal and equal spaced grid. These two properties are fundamental to get efficient implementation of low-level image processing operations such as Gaussian filtering and gradient computation.

The regularization procedure makes the surface area of each patch slightly bigger and hence, there are regions of the sphere that are covered by several patches. In order to avoid

artificial border artifacts and transfer information across patches, each patch is surrounded with a belt of interpolation coordinates to read image content from neighboring faces.

The processing of spherical images is divided in three main tasks. First, camera images are mapped onto each patch using the camera's calibration model. Second, low-level image subroutines run on each individual patch, using the interpolated image content in the border region. Finally, information in the overlapping areas is reconciled to create a uniform representation.

One of the great benefits of the Spherepix approach is the ability to apply any off-the-shelf algorithm to each individual patch and then interpret the results as quantities on the sphere. This is illustrated using the OpenCV implementations of SIFT feature point extraction and dense optical flow computation.

Chapter 4

Real-time Structure Flow

It always seems impossible until it's done.

Nelson Mandela

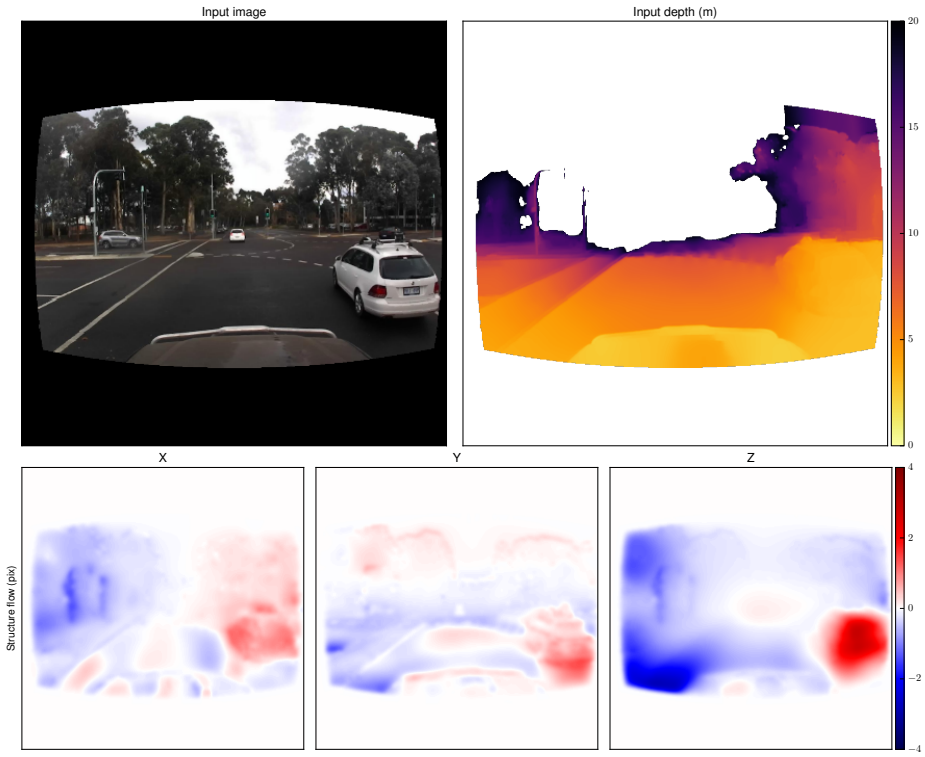


Figure 4.1: Structure flow is the vector field describing the 3D motion of the environment as seen from the camera. In this scene, the mounted camera moves forward in the z axis. The relative velocity of the static parts of the scene appears with a strong negative component in the z axis (blue). The vehicle on the right moves away from the camera and hence, its relative z velocity is positive. The residual x and y flow components are the result of fusing image data (inducing optical flow) and depth measurements.

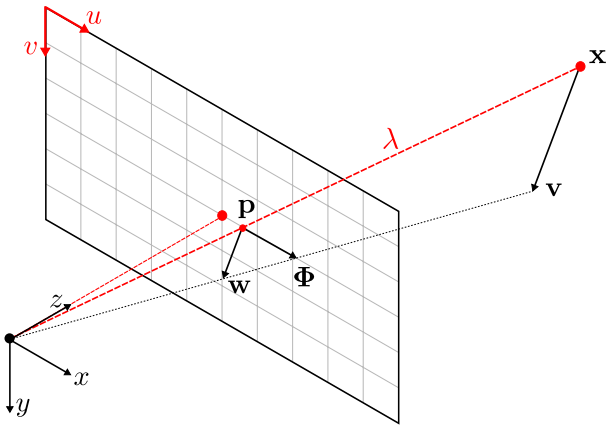
Local motion control of a robotic vehicle using visual remains a difficult task in robotics. Optical flow has been widely used to incorporate visual information into the robot's control algorithm. Applications such as landing and obstacle avoidance have successfully been explored in the past. Motion control algorithms that rely on raw optical flow are limited to specific scenarios, for example: grazing landing Srinivasan et al. [2000], corridor centering Srinivasan [2011b] and altitude regulation Ruffier and Franceschini [2005]. More sophisticated tasks such as landing and obstacle avoidance require post-processing of the optical flow field to infer the 'looming' effect, or optical flow divergence, that occurs when an object is approached directly. For example, Nelson and Aloimonos [1989] compute the divergence of the optical flow field and then use this for obstacle avoidance. Hamel and Mahony [2002], Herisse et al. [2008] and McCarthy and Barnes [2012] integrate the optical flow field divergence over the camera's field of view to compute the relative 3D velocity of the landing platform, effectively computing the looming effect by integration over a large area.

This chapter introduces a new robo-centric spatio-temporal representation of motion, the *structure flow* Adarve and Mahony [2016b]. Structure flow generalizes classical optic flow, that measures translation of an image point across the image surface, by including an additional normal component measuring the rate of angular 'looming' at each point in the image. Structure flow will have a similar utility to optic flow for control of robotic vehicles where in addition to existing methodologies, the normal component of structure flow directly yields motion cues associated with approach to obstacles. Geometrically. structure flow is a 3-vector assigned to each 'pixel' in the image comprised of the three-dimensional Euclidean velocity between the robot and the environment (the scene flow Vedula et al. [1999]) scaled by the inverse range of the scene.

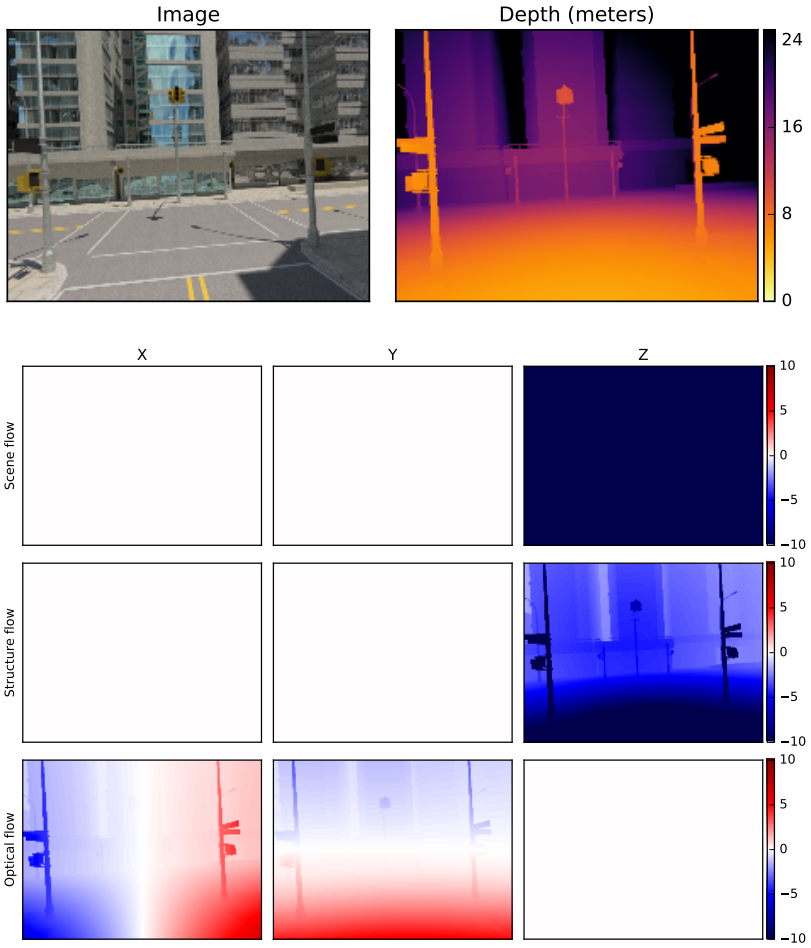
Figure 4.2b illustrates the optical, scene and structure flow fields for a perspective camera moving at 10ms^{-1} in collision course with a building located approximately at 15m. Since the scene is static, the calculated scene flow is equal for all pixels in the image, the negative of the camera velocity. Scaling the scene flow by the inverse of the depth field, one obtains the structure flow field. Note the clear identification between objects close to or far away to the camera that is of direct importance in vehicle control. In contrast, the optical flow is a divergent vector field with the focus of expansion located in the center of the image; the direction of motion. Although the vehicle is moving fast, the optical flow in the central region is small, and it is difficult to evaluate the time to contact for the vehicle to collide with the building.

In addition to its relevance for robotic motion control, structure flow plays an intrinsic role in understanding image kinematics and dynamics. Image kinematics are governed by the well known constant brightness condition Barron [1994] which depends on optical flow, or in the general case, on the projection of structure flow on the image plane. For reasonable assumptions on the environment and camera motion, Partial Differential Equation (PDE) can be derived for the evolution of the structure flow that depends only on exogenous acceleration and rotation of the camera. These PDEs are naturally derived using a spherical camera model, and as it will be shown, they can efficiently been implemented using the Spherepix data structured developed in Chapter 3.

This chapter also introduces a filtering algorithm to compute structure flow in real-time from stereo image data. The approach taken is a highly distributed predictor-update algorithm



(a) Scene $\mathbf{v} \in \mathbb{R}^3$, structure $\mathbf{w} \in \mathbb{R}^3$ and optical flow $\mathbf{\Phi} \in \mathbb{R}^2$ vectors for a point $\mathbf{x}$ in the scene. Point $\mathbf{x}$ moves with a relative velocity (scene flow) $\mathbf{v}$ with respect to the camera body fixed frame, and projects to pixel $\mathbf{p}$ on the image plane. The optical flow $\mathbf{\Phi}$ at $\mathbf{p}$ is a two-dimensional projection of scene flow on the image plane. The structure flow vector $\mathbf{w}$ is the scaling of the scene flow by the inverse of the distance λ to $\mathbf{x}$.



(b) Scene, structure and optical flow fields for a perspective camera mounted on a forward moving vehicle at 10 $^m/_s$. The simulated perspective camera runs at 100 Hz.

Figure 4.2: Scene, optical and structure flow fields for a perspective.

implemented on GPU hardware. The prediction step is a numerical implementation of a PDE integration scheme that propagates the current estimate of the structure flow forward in time. Error in the structure flow estimate can then be estimated across a sequence of temporal frames. A simple least-squares regularized update is used to iteratively correct errors in the structure flow estimate. The resulting algorithm runs on a Nvidia GTX-780 GPU, processing 512×512 at approximately 600 Hz for flow vectors up to 8 pixels in magnitude. The relative performance of the algorithm far outperforms (10-20 times faster) all scene flow algorithms in the literature in terms of processing speed. Although it is less accurate than some classical algorithms, its advantages in speed, dense estimation, and robustness make it highly suitable for real-world mobile robotic applications.

4.1 Outline

Section 4.2 provides the fundamental concepts to understand structure flow and the relevant literature. Section 4.3 goes into the details of the structure flow field on spherical cameras. Section 4.4 defines the partial differential equations modeling spatio-temporal evolution of image, structure and structure flow on the sphere. Section 4.5 describes the proposed filtering algorithm to estimate structure flow in real time. Section 4.6 provides the numerical details for implementing the PDEs in Section 4.4. Section 4.7 gives the experimental evaluation of the structure flow filtering algorithm on ground truth and real-life data. Finally, the chapter is closed with some summary remarks in Section 4.8.

4.2 Background

4.2.1 Kinematics

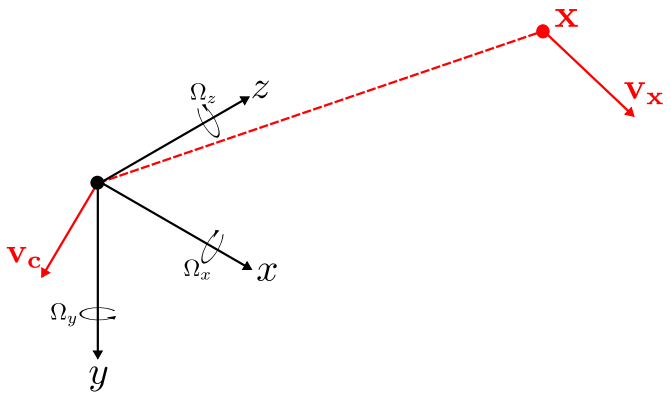


Figure 4.3: Kinematics of point $\mathbf{x}$ expressed in the camera body fixed frame.

Consider Figure 4.3. Let $\mathbf{v}_c$ and $\boldsymbol{a}_c$ denote the linear velocity and acceleration respectively of the camera, taken with respect to an *inertial frame*, and expressed in the camera frame. Let Ω denote the angular velocity of the camera expressed in the camera frame and let $\Omega_\times$ be the skew-symmetric matrix such that $\Omega \times v = \Omega_\times v$ for any vector v. For the present derivation,

it is assumed that Ω and $\boldsymbol{a}_c$ are approximately constant. This is a reasonable assumption given the very high frame rates of the algorithms proposed in this chapter. With this assumption, the second order kinematics of the camera are

$$\frac{\mathrm{d}\mathbf{v}_c}{\mathrm{d}t} = -\Omega_\times \mathbf{v}_c + \boldsymbol{a}_c \tag{4.1}$$

$$\frac{\mathrm{d}\Omega}{\mathrm{d}t} = 0 \tag{4.2}$$

Both Ω and $\boldsymbol{a}_c$ are readily available from a typical inertial measurement unit and they are assumed to be known. Notice that the direction of the linear velocity vector is affected by the rotation of the camera body fixed frame through the matrix multiplication $-\Omega_\times \mathbf{v}_c$.

Let $\mathbf{x} = \mathbf{x}(t) \in \mathbb{R}^3$ denote the position of a point in the scene at time t in body fixed frame (Figure 4.3). The distance or depth of point $\mathbf{x}(t)$ to the origin of the camera is

$$\lambda(t) = \langle \mathbf{x}(t), \mathbf{x}(t) \rangle^{1/2} \tag{4.3}$$

Let $\mathbf{v}_\mathbf{x}$ denote the velocity of the point $\mathbf{x}$ with respect to the *inertial frame* and expressed in the camera body fixed frame. It is assumed that points in the scene are moving with a constant velocity with respect to the inertial frame. With these assumptions, the first and second order kinematics of a point $\mathbf{x}$ in the scene expressed in the camera frame are:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = -\Omega_\times \mathbf{x} + \mathbf{v}_\mathbf{x} - \mathbf{v}_c \tag{4.4}$$

$$\frac{\mathrm{d}\mathbf{v}_\mathbf{x}}{\mathrm{d}t} = -\Omega_\times \mathbf{v}_\mathbf{x} \tag{4.5}$$

These equations will be used to derive the scene, optical and structure flow in the next sections.

4.2.2 Scene and optical flow on the image plane

Consider a perspective camera as the one in Figure 4.2a. The camera is characterized by the intrinsics matrix K and the focal point is located at the origin of the camera body fixed frame. Consider the kinematic equations in Section 4.2.1. A point $\mathbf{x}(t)$ is projected to pixel position $\mathbf{p} \in \mathbb{R}^2$ in the camera image plane as

$$\begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} = \frac{K\mathbf{x}}{\langle e_3, \mathbf{x} \rangle} \tag{4.6}$$

where $e_3 = \begin{pmatrix} 0, 0, 1 \end{pmatrix}^\top$ and $\begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix}$ is the pixel coordinate expressed in homogeneous coordinates.

The scene flow field, Vedula et al. [1999, 2005], is the vector field $\mathbf{v} : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^3$ that assigns the three dimensional velocity of the scene relative to the camera at time t (Equation (4.4)) to each pixel $\mathbf{p}$ in the image. That is,

$$\mathbf{v}(\mathbf{p}, t) = -\Omega_\times \mathbf{x} + \mathbf{v}_\mathbf{x} - \mathbf{v}_c \tag{4.7}$$

Notice that the right hand side of Equation (4.7) does not depend on the perspective camera geometry. The scene flow is completely characterized by the kinematics of the camera and the scene.

The optical flow field $\mathbf{\Phi} : \mathbb{R}^2 \times \mathbb{R} \rightarrow \mathbb{R}^2$ is the projection of the scene flow into the image plane of the camera.

$$\begin{pmatrix} \mathbf{\Phi}(\mathbf{p}, t) \\ 0 \end{pmatrix} = \frac{1}{\langle e_3, \mathbf{x} \rangle} \left[K \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} - \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} \langle e_3, \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \rangle \right] \tag{4.8}$$

The derivation of Equation (4.8) is provided in Appendix A. Contrary to scene flow, the computation of optical flow depends on the camera geometry (intrinsics matrix K) and the actual pixel position in the image.

4.2.3 Estimation of scene flow

This section reviews the relevant literature for estimating scene flow. Algorithms are classified according to the time of input measurements used for extracting the scene flow field.

Stereoscopic approaches

Stereoscopic algorithms use image sequences captured using a stereo camera to estimate the 3D scene flow. Early work by Patras et al. [1996] formulates the joint estimation of disparity and motion (scene flow) from stereo sequences. Huguet and Devernay Huguet and Devernay [2007] proposed a variational approach for the joint computation of disparity and optical flow. This formulation of optical flow plus temporal disparity change is known as *disparity flow* Gong and Yang [2006]. Wedel et al. [2011] decouple the variational formulation of to compute flow and disparity in two separate stages while preserving stereo constraints. Vogel et al. [2015] compute scene flow by means of a piece-wise planar segmentation of stereo images for which a 3D rotation and translation (scene flow) is computed. The accuracy of recent two-pair stereo based algorithms is compiled on the Kitti scene flow dataset by Menze and Geiger [2015].

RGB + Depth approaches

Algorithms using RGB-D sensors to compute scene flow combine monocular images and depth measurements to estimate the 3D velocity field. Letouzey et al. [2011] combine sparse features points and dense smoothness constraints to estimate dense scene flow from RGB-D data. Hadfield and Bowden [2011] use a particle filter formulation to track points in the scene. Quiroga et al. [2012] use brightness and depth data in a Lukas-Kanade type tracking framework to compute scene flow parametrized as a rotation plus translation transform.

Estimation from depth measurements

If only depth/range measurements are available it is still possible to compute *range flow*. Yamamoto *et. al.* derived the range flow constraint Yamamoto et al. [1993], a partial differential equation to model the change of depth. Differential methods to compute range flow suffer from a 3D version of the aperture problem similar to that found in optical flow methods Spies et al.

[2002]. Herbst et al. [2013] use this differential constraint on depth and combined with color image to derive a variational method to compute scene flow following a formulation similar to Brox et al. [2004] for computing optical flow.

4.2.4 Real-time algorithms

In terms of real-time capable dense 3D motion estimation algorithms, most methods use Graphics Processing Units (GPU) for implementing per-pixel level operations in parallel. Gong [2009] reports 12 Hz optical and disparity flow estimation on QVGA image resolution (320×240). Rabe et al. [2010] reported 25 Hz frequency on VGA (640×480) image sequences using GPU. The algorithm of Wedel et al. [2011] reports frame rates of 20 Hz also at QVGA resolution. RGB-D flow by Herbst et al. [2013] runs between 8-30 Hz at QVGA depending on the amount of smoothing required.

4.3 Optical, Scene and Structure Flow on the Sphere

Recall the kinematic equations derived in Section 4.2.1 and consider Figure 4.4 where a spherical camera is located in the center of the body-fixed frame. The environment is parametrized by directions $\boldsymbol{\eta} \in S^2$ in the sphere. The tangent space $T_\eta S^2$ associated to each point $\boldsymbol{\eta}$ is defined as (Chapter 3)

$$T_\eta S^2 = \{\boldsymbol{\mu} \in \mathbb{R}^3 \mid \langle \boldsymbol{\eta}, \boldsymbol{\mu} \rangle = 0\} \tag{4.9}$$

and the projection matrix $\mathbf{P}_\eta : \mathbb{R}^3 \to T_\eta S^2$ is the orthogonal projection onto $T_\eta S^2$

$$\mathbf{P}_\eta = (\mathbf{I}_3 - \boldsymbol{\eta}\boldsymbol{\eta}^\top) \tag{4.10}$$

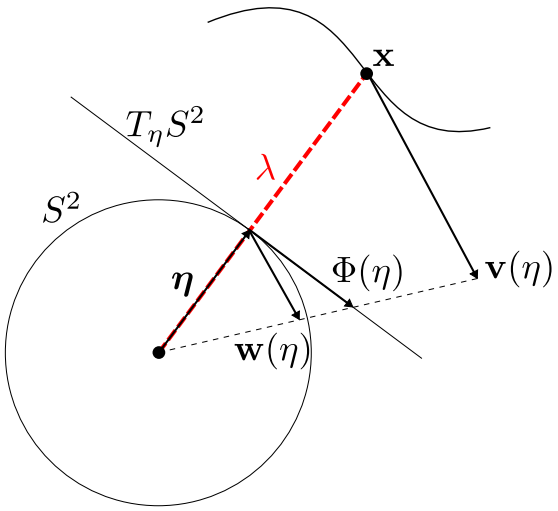


Figure 4.4: Scene $\mathbf{v}(\boldsymbol{\eta})$, structure $\mathbf{w}(\boldsymbol{\eta})$ and optical flow $\boldsymbol{\Phi}(\boldsymbol{\eta})$ for a spherical camera.

The first visible point in the environment in direction $\boldsymbol{\eta}$ at time t is denoted $\mathbf{x}(\boldsymbol{\eta}, t)$. The

range or depth map around the robot is a single valued depth field $\lambda : S^2 \times \mathbb{R} \to \mathbb{R}$,

$$\lambda(\boldsymbol{\eta}, t) = \langle \mathbf{x}(\boldsymbol{\eta}, t), \mathbf{x}(\boldsymbol{\eta}, t) \rangle^{1/2}. \tag{4.11}$$

Although range $\lambda(\boldsymbol{\eta}, t)$ appears to be the more intuitive variable, it turns out that it is more natural in the context of the mathematical development to use an inverse ratio of range $\rho : S^2 \times \mathbb{R} \to \mathbb{R}$,

$$\rho(\boldsymbol{\eta}, t) = \frac{\lambda_{\text{ref}}}{\lambda(\boldsymbol{\eta}, t)}, \tag{4.12}$$

where λ_{ref} is the reference range to the spherical camera. The value $\lambda_{\text{ref}} = 1\text{m}$ is chosen for simplicity and suppresses the λ_{ref} notation in the remainder of the chapter. However, one should still think of $\rho(\boldsymbol{\eta}, t)$ as a dimensionless ratio of ranges related to the perceived visual angle of an object at range $\lambda(\boldsymbol{\eta}, t)$ as observed on the sphere of radius λ_{ref}. In particular, $\rho(\boldsymbol{\eta}, t)$ is intrinsically a visual measure or angle, whereas $\lambda(\boldsymbol{\eta}, t)$ is extrinsically related to the physical scene and has units of meters.

The scene flow is defined as the 3D velocity of the environment relative to the camera Vedula et al. [1999] of a point $\mathbf{x}(\boldsymbol{\eta}, t)$ in the scene. To compute scene flow, consider a solution $\mathbf{x}(\tau)$ of (4.4) for $\tau \in (t - \delta, t + \delta)$, $\delta > 0$, in a small time interval around an 'initial condition' $\mathbf{x}(t) = \mathbf{x}(\boldsymbol{\eta}, t)$ in the desired direction $\boldsymbol{\eta}$ at time t. Choose $\boldsymbol{\eta}(\tau)$ to evolve such that $\mathbf{x}(\tau) = \mathbf{x}(\boldsymbol{\eta}(\tau), \tau)$ is the point in the scene associated with the evolution of $\mathbf{x}(\tau)$ and note that $\boldsymbol{\eta}(t) = \boldsymbol{\eta}$; that is $\boldsymbol{\eta}(\tau)$ evaluated at $\tau = t$ is $\boldsymbol{\eta}$. The scene flow is the vector field $\mathbf{v} : S^2 \times \mathbb{R} \to \mathbb{R}^3$ defined as[1]

$$\mathbf{v}(\boldsymbol{\eta}, t) = \left. \frac{\mathrm{d}\mathbf{x}(\boldsymbol{\eta}(\tau), \tau)}{\mathrm{d}\tau} \right|_{\tau = t} = -\Omega_{\times}\mathbf{x} + \mathbf{v}_{\mathbf{x}} - \mathbf{v}_c. \tag{4.13}$$

It is important to note that, whereas (4.4) tracks a particle in the environment, the scene-flow is a vector field that assigns values for all points parameterized on the sphere. In particular, $\boldsymbol{\eta}$ and t are independent variables in the expression (4.13) for $\mathbf{v}(\boldsymbol{\eta}, t)$.

Define the structure flow field $\mathbf{w} : S^2 \times \mathbb{R} \to \mathbb{R}^3$ as the three-dimensional vector field consisting of the scene flow scaled by the inverse depth

$$\begin{aligned} \mathbf{w}(\boldsymbol{\eta}, t) &= \frac{1}{\lambda(\boldsymbol{\eta}, t)} \mathbf{v}(\boldsymbol{\eta}, t) \\ &= -\Omega_{\times}\boldsymbol{\eta} + \frac{1}{\lambda(\boldsymbol{\eta}, t)}(\mathbf{v}_{\mathbf{x}} - \mathbf{v}_c) \end{aligned} \tag{4.14}$$

Note that while scene flow has units of m.s^{-1}, structure flow has units of rad.s^{-1}, where the angle is related to the ratio of distances, as would be expected of an image based flow measure. The following notation is used to decompose structure flow into rotational and stabilized

[1] A more common notation in the literature used for the total derivative is

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}(t) = \left. \frac{\mathrm{d}\mathbf{x}(\boldsymbol{\eta}(\tau), \tau)}{\mathrm{d}\tau} \right|_{\tau = t}$$

where the underlying construction of the segment of trajectory $\mathbf{x}(\tau)$ is understood. A more explicit notation is chosen to provide more clarity in the derivations.

(linear) components, $\mathbf{w} = \mathbf{w}_r + \mathbf{w}_s$, where

$$\mathbf{w}_r(\boldsymbol{\eta}, t) = -\Omega_\times \boldsymbol{\eta}, \tag{4.15}$$

$$\mathbf{w}_s(\boldsymbol{\eta}, t) = \frac{1}{\lambda(\boldsymbol{\eta}, t)}(\mathbf{v}_\mathbf{x} - \mathbf{v}_c). \tag{4.16}$$

In order to compute evolution equations for structure flow it is necessary to compute the total time derivative of the flow. First, the total derivatives are computed for $\boldsymbol{\eta}(\tau)$ and $\lambda(\tau)$, defined along trajectories $\mathbf{x}(\tau) = \mathbf{x}(\boldsymbol{\eta}(\tau), \tau)$ induced by the equations of motion (4.1), (4.2), (4.4) and (4.5).

For depth $\lambda(\tau)$, one has

$$\begin{aligned}
\left.\frac{\mathrm{d}\lambda(\tau)}{\mathrm{d}\tau}\right|_{\tau=t} &= \left.\frac{\mathrm{d}}{\mathrm{d}\tau}\langle \mathbf{x}(\tau), \mathbf{x}(\tau)\rangle^{1/2}\right|_{\tau=t} \\
&= \left.\frac{\mathbf{x}(\tau)^\top}{\langle \mathbf{x}(\tau), \mathbf{x}(\tau)\rangle^{1/2}} \frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau}\right|_{\tau=t} \\
&= \left.\langle \boldsymbol{\eta}(\tau), \frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau}\rangle\right|_{\tau=t} \\
&= \left.\frac{\lambda(\boldsymbol{\eta}, \tau)}{\lambda(\boldsymbol{\eta}, \tau)}\langle \boldsymbol{\eta}(\tau), \mathbf{v}(\boldsymbol{\eta}, \tau)\rangle\right|_{\tau=t} \\
&= \lambda(\boldsymbol{\eta}, t)\langle \boldsymbol{\eta}, \mathbf{w}(\boldsymbol{\eta}, t)\rangle
\end{aligned} \tag{4.17}$$

where the last line follows by substituting for structure flow (4.14) at $\tau = t$. A similar derivation for the inverse depth $\rho(t)$ yields

$$\left.\frac{\mathrm{d}\rho(\tau)}{\mathrm{d}\tau}\right|_{\tau=t} = -\rho(\boldsymbol{\eta}, t)\langle \boldsymbol{\eta}, \mathbf{w}(\boldsymbol{\eta}, t)\rangle \tag{4.18}$$

We note that the right hand side of both (4.17) and (4.18) depend only on the field variables $\mathbf{w}$, λ and ρ defined at $(\boldsymbol{\eta}, t)$ and not on the trajectory $\mathbf{x}(\tau)$ from which they are derived.

Next, consider the rate of change of the direction $\boldsymbol{\eta}(\tau) = {}^{\mathbf{x}(\tau)}/_{\lambda(\tau)}$. One has

$$\begin{aligned}
\left.\frac{\mathrm{d}\boldsymbol{\eta}(\tau)}{\mathrm{d}\tau}\right|_{\tau=t} &= \left.\frac{\mathrm{d}}{\mathrm{d}\tau}\left(\frac{\mathbf{x}(\tau)}{\lambda(\tau)}\right)\right|_{\tau=t} \\
&= \left.\frac{\frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau}\lambda(\tau) - \mathbf{x}(\tau)\frac{\mathrm{d}\lambda(\tau)}{\mathrm{d}\tau}}{\lambda(\tau)^2}\right|_{\tau=t} \\
&= \left.\frac{1}{\lambda(\tau)}\left(\frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau} - \boldsymbol{\eta}(\tau)\langle \boldsymbol{\eta}(\tau), \frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\tau}\rangle\right)\right|_{\tau=t} \\
&= \frac{1}{\lambda(\boldsymbol{\eta}, t)}(\mathbf{I} - \boldsymbol{\eta}\boldsymbol{\eta}^\top)\mathbf{v}(\boldsymbol{\eta}, t) \\
&= \mathbf{P}_\eta \mathbf{w}(\boldsymbol{\eta}, t)
\end{aligned} \tag{4.19}$$

Note that $\mathbf{P}_\eta \mathbf{w}$ satisfies $\langle \boldsymbol{\eta}, \mathbf{P}_\eta \mathbf{w}\rangle = 0$ and hence, is an element of the tangent space of $\boldsymbol{\eta}$. Indeed, the total time derivative of $\boldsymbol{\eta}(\tau)$ is the optical flow $\boldsymbol{\Phi}(\boldsymbol{\eta}, t) = \mathbf{P}_\eta \mathbf{w}$ perceived by the

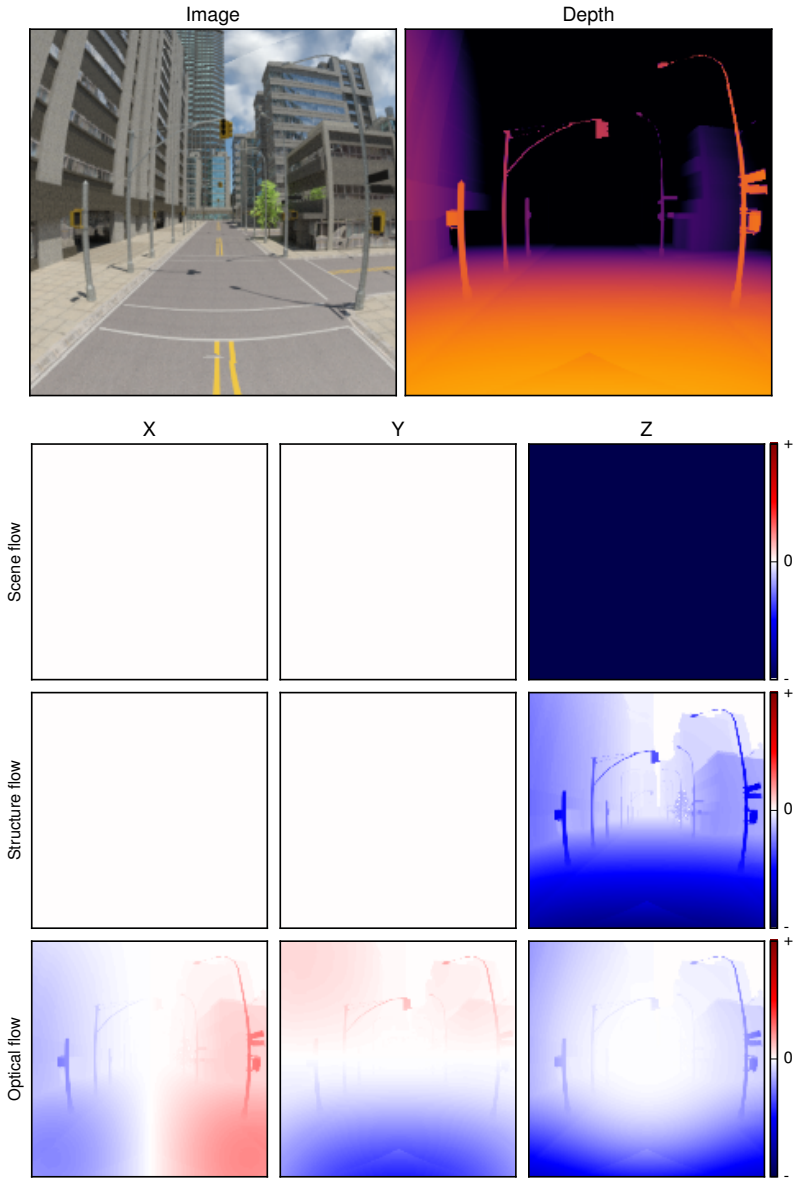


Figure 4.5: Scene, structure and optical flow fields on for a forward moving camera in a static scene. Since the scene is static, the scene flow equals $-\mathbf{v}_c$ for all pixels and in particular, the xy components of scene flow are zero. The structure flow combines the scene flow and the inverse depth at each pixel to create a velocity field that differentiates between close and distant objects. In contrast, the optical flow generates a divergent velocity field product of the projection onto tangent space. The focus of expansion of the optical flow at the image center corresponds to the direction of motion. In particular, notice that the optical flow field has a non-zero z value around the border region of the image. This is a consequence of considering optical flow as a vector field in the tangent space of the sphere rather than on a perspective image plane.

spherical camera Koenderink and van Doorn [1987]; Hamel and Mahony [2002].

Figure 4.5 illustrates the xyz components of scene, structure and optical flow fields for a static scene and a forward moving camera (along the z direction) with no rotation.

Finally, consider the total derivative of the structure flow vector $\mathbf{w}(\tau) = \mathbf{w}(\boldsymbol{\eta}(\tau), \tau)$ taken along a trajectory induced by the equations of motion (4.1), (4.2), (4.4) and (4.5). One has

$$\begin{aligned}
\left.\frac{\mathrm{d}\mathbf{w}(\tau)}{\mathrm{d}\tau}\right|_{\tau=t} &= \left.\frac{\mathrm{d}}{\mathrm{d}\tau}\left(-\Omega_\times \boldsymbol{\eta}(\tau) + \frac{1}{\lambda(\tau)}(\mathbf{v}_\mathbf{x}(\tau) - \mathbf{v}_c(\tau))\right)\right|_{\tau=t} \\
&= -\left.\frac{\mathrm{d}}{\mathrm{d}\tau}\left(\Omega_\times \boldsymbol{\eta}(\tau)\right)\right|_{\tau=t} + \left.\frac{\mathrm{d}\rho(\tau)}{\mathrm{d}\tau}(\mathbf{v}_\mathbf{x} - \mathbf{v}_c)\right|_{\tau=t} + \rho(\tau)\left.\frac{\mathrm{d}}{\mathrm{d}\tau}(\mathbf{v}_\mathbf{x}(\tau) - \mathbf{v}_c(\tau))\right|_{\tau=t} \\
&= -\Omega_\times \mathbf{P}_\eta \mathbf{w}(\boldsymbol{\eta}, t) - \rho(\boldsymbol{\eta})\langle \boldsymbol{\eta}, \mathbf{w}(\boldsymbol{\eta}, t)\rangle(\mathbf{v}_\mathbf{x}(t) - \mathbf{v}_c(t)) \\
&\qquad - \rho(\boldsymbol{\eta})\Omega_\times(\mathbf{v}_\mathbf{x}(t) - \mathbf{v}_c(t)) + \rho(\boldsymbol{\eta})\boldsymbol{a}_c \\
&= -\Omega_\times \mathbf{P}_\eta \mathbf{w}(\boldsymbol{\eta}, t) - \mathbf{w}_s(\boldsymbol{\eta}, t)\langle \boldsymbol{\eta}, \mathbf{w}(\boldsymbol{\eta}, t)\rangle - \Omega_\times \mathbf{w}_s(\boldsymbol{\eta}, t) + \rho(\boldsymbol{\eta})\boldsymbol{a}_c
\end{aligned} \tag{4.20}$$

Define the grouped acceleration term

$$\boldsymbol{a}_\mathbf{w}(\boldsymbol{\eta}, t) = \rho \boldsymbol{a}_c - \Omega_\times \mathbf{w}_s(\boldsymbol{\eta}, t) = \rho \boldsymbol{a}_c - \Omega_\times\left(\mathbf{w} + \Omega_\times \boldsymbol{\eta}\right) \tag{4.21}$$

This is the exogenous linear acceleration of the structure flow field at $(\boldsymbol{\eta}, t)$ due to camera motion along with the coriolis term $\Omega_\times \mathbf{w}_s(\boldsymbol{\eta}, t)$ associated with the fact that the linear velocities are expressed in a rotating frame. Recall that $\mathbf{w}_s = \mathbf{w} - \mathbf{w}_r$ and $\mathbf{w}_r = -\Omega_\times \boldsymbol{\eta}$. Rewriting in terms of the full structure flow and $\boldsymbol{a}_\mathbf{w}$ one has

$$\begin{aligned}
\left.\frac{\mathrm{d}\mathbf{w}(\tau)}{\mathrm{d}\tau}\right|_{\tau=t} &= -\Omega_\times \mathbf{P}_\eta \mathbf{w}(\boldsymbol{\eta}, t) + \Omega_\times \boldsymbol{\eta}\langle \boldsymbol{\eta}, \mathbf{w}(\boldsymbol{\eta}, t)\rangle - \mathbf{w}(\boldsymbol{\eta}, t)\langle \boldsymbol{\eta}, \mathbf{w}(\boldsymbol{\eta}, t)\rangle + \boldsymbol{a}_\mathbf{w}(\boldsymbol{\eta}, t) \\
&= -\mathbf{w}(\boldsymbol{\eta}, t)\langle \boldsymbol{\eta}, \mathbf{w}(\boldsymbol{\eta}, t)\rangle - \Omega_\times \mathbf{w}(\boldsymbol{\eta}, t) + \boldsymbol{a}_\mathbf{w}(\boldsymbol{\eta}, t)
\end{aligned} \tag{4.22}$$

Here, the term $-\Omega_\times \mathbf{w}$ is to be expected since $\mathbf{w}$ is expressed with respect to the rotating camera frame. The driving term $-\mathbf{w}\langle \boldsymbol{\eta}, \mathbf{w}\rangle$ is the autonomous growth or decrease in $\mathbf{w}$ due to its range velocity component and the field acceleration $\boldsymbol{a}_\mathbf{w}$ defined earlier.

4.4 Evolution equations

This section derives the fundamental partial differential equations that model the temporal change of image brightness and structure flow on the spherical camera.

4.4.0.1 Image brightness

Let $Y(\boldsymbol{\eta}, t)$ denote the image brightness in the direction $\boldsymbol{\eta} \in S^2$ at time t associated to point $\mathbf{x}(t)$ in the scene. That is $Y : S^2 \times \mathbb{R} \to \mathbb{R}$ is modeled as a scalar field on the sphere. A common assumption on differential methods for computing either optical or scene flow is that image brightness is constant and the surface is Lambertian. That is, the brightness value corresponding to point $\mathbf{x}(t)$ as seen by the camera does not change with the evolution of $\mathbf{x}(t)$ in time. This assumption correspond to the well known constant brightness condition on the

total derivative of the image brightness field Barron [1994],

$$\frac{\mathrm{d}Y}{\mathrm{d}t}(t) = \left.\frac{\mathrm{d}Y(\boldsymbol{\eta}(\tau))}{\mathrm{d}\tau}\right|_{\tau=t} = 0. \tag{4.23}$$

Expressing the total derivative of $Y(\boldsymbol{\eta}, t)$ in terms of its partial derivatives one obtains

$$\frac{\partial Y}{\partial \boldsymbol{\eta}}\frac{\mathrm{d}\boldsymbol{\eta}}{\mathrm{d}t} + \frac{\partial Y}{\partial t} = \frac{\mathrm{d}Y}{\mathrm{d}t} \tag{4.24}$$

Here $\frac{\partial Y}{\partial \boldsymbol{\eta}} \in T_\eta S^2$ denotes the image gradient and it is an element of the tangent space of $\boldsymbol{\eta}$. Moreover, $\frac{\mathrm{d}\boldsymbol{\eta}}{\mathrm{d}t} = \left.\frac{\mathrm{d}\boldsymbol{\eta}(\tau)}{\mathrm{d}\tau}\right|_{\tau=t} = \boldsymbol{\Phi}(\boldsymbol{\eta}, t)$ is the optical flow vector at $\boldsymbol{\eta}$ defined in Equation (4.19). Expressing PDE (4.24) in terms of structure flow, one has

$$\frac{\partial Y}{\partial t} = -\frac{\partial Y}{\partial \boldsymbol{\eta}}\mathbf{P}_\eta \mathbf{w} \tag{4.25}$$

Equation (4.25) is the classical brightness conservation equation Barron [1994] expressed in spherical camera coordinates.

4.4.0.2 Structure flow

Analogous to the brightness conservation, consider $\mathbf{w}(\boldsymbol{\eta}, t)$ as a continuous vector field on the sphere. The relative change with respect to $\boldsymbol{\eta}$ and t is

$$\frac{\partial \mathbf{w}}{\partial \boldsymbol{\eta}}\frac{\mathrm{d}\boldsymbol{\eta}}{\mathrm{d}t} + \frac{\partial \mathbf{w}}{\partial t} = \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} \tag{4.26}$$

where $\frac{\partial \mathbf{w}}{\partial \boldsymbol{\eta}}$ is the Jacobian matrix of $\mathbf{w}$ at $\boldsymbol{\eta}$ and $\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = \left.\frac{\mathrm{d}\mathbf{w}(\boldsymbol{\eta}(\tau),\tau)}{\mathrm{d}\tau}\right|_{\tau=t}$ as derived in (4.22). In this case, the total flow is not conserved, that is, $\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} \neq 0$, and one must substitute (4.22) to derive the associated PDE

$$\frac{\partial \mathbf{w}}{\partial t} = -\frac{\partial \mathbf{w}}{\partial \boldsymbol{\eta}}\mathbf{P}_\eta \mathbf{w} - \mathbf{w}\langle \boldsymbol{\eta}, \mathbf{w}\rangle - \Omega_\times \mathbf{w} + \boldsymbol{a}_\mathbf{w} \tag{4.27}$$

Equation (4.27) describes a non-linear transport process where the structure flow is propagated on the sphere at a velocity equal to the induced optical flow $\mathbf{P}_\eta \mathbf{w}$. Additionally, the source terms inject or remove energy from the structure flow field according to the kinematics of the robot and the scene relative to it.

4.4.0.3 Depth and inverse depth

Similar partial differential equations can be derived for the conservation of the depth field $\lambda(\boldsymbol{\eta}, t)$ and the inverse depth $\rho(\boldsymbol{\eta}, t)$. Considering the total derivatives of depth and inverse

depth developed in Equations (4.17) and (4.18), one obtains:

$$\frac{\partial \lambda}{\partial t} = -\frac{\partial \lambda}{\partial \boldsymbol{\eta}} \mathbf{P}_{\eta} \mathbf{w} + \lambda \langle \boldsymbol{\eta}, \mathbf{w} \rangle \tag{4.28}$$

$$\frac{\partial \rho}{\partial t} = -\frac{\partial \rho}{\partial \boldsymbol{\eta}} \mathbf{P}_{\eta} \mathbf{w} - \rho \langle \boldsymbol{\eta}, \mathbf{w} \rangle \tag{4.29}$$

The conservation of inverse depth has previously been reported in the literature of depth reconstruction from motion Bonnabel and Rouchon [2009]; Zarrouati et al. [2012]. In practice, Equation (4.29) is preferred since ρ is a dimensionless ratio (4.12) that has an intrinsic interpretation as visual measure. Note also that (4.29) has the same structure as the first two terms of (4.27) which leads to desirable computational properties of the combined system of equations considered in Section 4.5.

4.5 Filter Algorithm

This section introduces a filtering approach for the computation of structure flow in real time using brightness and depth measurements. Similar to the optical flow filter described in Chapter 2, a pyramid structure is used to support large pixel displacements between consecutive images. Let H denote the number of levels in the pyramid structure indexed as $h = 1, \ldots, H$, where $h = 1$ denotes the original resolution level.

The filter state at discrete time index k is denoted by the set

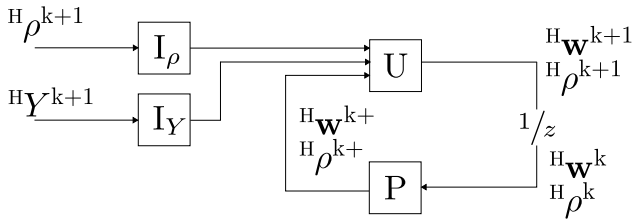
$$X^k = \left\{ \begin{Bmatrix} {}^{1}\Delta\mathbf{w}^{\mathrm{k}} \\ {}^{1}\rho^{\mathrm{k}} \end{Bmatrix}, \begin{Bmatrix} {}^{2}\Delta\mathbf{w}^{\mathrm{k}} \\ {}^{2}\rho^{\mathrm{k}} \end{Bmatrix}, \ldots, \begin{Bmatrix} {}^{\mathrm{H}}\mathbf{w}^{\mathrm{k}} \\ {}^{\mathrm{H}}\rho^{\mathrm{k}} \end{Bmatrix} \right\} \tag{4.30}$$

Each level h contains an estimate of the inverse depth field ρ_k^{h} using data of the corresponding pyramid level. State ${}^{\mathrm{H}}\mathbf{w}^{\mathrm{k}}$ at top level H represents a coarse estimate of the structure flow based on low resolution down-sampled data. The flow at each level, h, of the pyramid is denoted ${}^{\mathrm{h}}\mathbf{w}^{\mathrm{k}}$, however, the flow itself is not used as the dynamic state of the filter for the lower levels of the pyramid. Instead, the lower level filter states ${}^{\mathrm{h}}\Delta\mathbf{w}^{\mathrm{k}}$ define the increment to the flow ${}^{\mathrm{h+1}}\mathbf{w}^{\mathrm{k}}$ given higher resolution data at level h and the the structure flow is reconstructed by

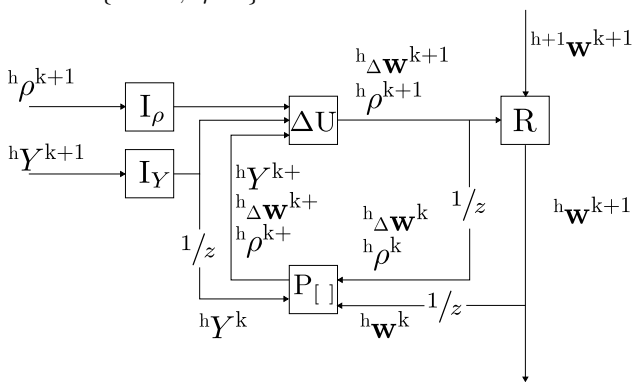
$${}^{\mathrm{h}}\mathbf{w}^{\mathrm{k}} = {}^{\mathrm{h+1:h}}\mathbf{w}^{\mathrm{k}} + {}^{\mathrm{h}}\Delta\mathbf{w}^{\mathrm{k}} \tag{4.31}$$

applied recursively for $h = H - 1, \ldots, 1$. Here ${}^{\mathrm{h+1:h}}\mathbf{w}^{\mathrm{k}}$ is the flow at level $h + 1$ up-sampled to level h.

Figure 4.6a illustrates the filter structure at top level H. Blocks $[I_Y]$ and $[I_\rho]$ extract linear model parameters from image and inverse depth raw measurement data. Details of these processes are provided in Sections 4.5.1 and 4.5.2. In the prediction block $[P]$, current estimate ${}^{\mathrm{H}}\mathbf{w}^{\mathrm{k}}$ is used to create a prediction $\{{}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+}}, {}^{\mathrm{H}}\rho^{\mathrm{k+}}\}$ for the flow and inverse depth state at time $k+1$. Notation $k+$ is used to refer to the estate before new measurements at time $k+1$ are incorporated into it. The update block $[U]$, Section 4.5.4, takes the predicted state $\{{}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+}}, {}^{\mathrm{H}}\rho^{\mathrm{k+}}\}$ and brightness and inverse depth parameters at $k+1$ to create a new estimate of structure flow $\{{}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}}, {}^{\mathrm{H}}\rho^{\mathrm{k+1}}\}$.



(a) Top level filter loop. The structure flow $^{H}\mathbf{w}^{k}$ is used in the prediction stage $[P]$ to compute predictions $\{^{H}\mathbf{w}^{k+}, {}^{H}\rho^{k+}\}$. New raw image and inverse depth measurements are processed at $[I_Y]$ and $[I_\rho]$ blocks to extract model parameters. These parameters are used in the update stage $[U]$ to compute new state estimate $\{^{H}\mathbf{w}^{k+1}, {}^{H}\rho^{k+1}\}$.



(b) Lower levels filter loop. The reconstruction block $[R]$ reconstructs the flow at this level using the flow from level $h+1$ and state $^{h}{}_{\Delta}\mathbf{w}^{k+1}$. The reconstructed flow is used in the prediction stage $[P_{[]}]$ to create predictions $^{h}Y^{k+}, \{^{h}{}_{\Delta}\mathbf{w}^{k+}, {}^{h}\rho^{k+}\}$ and $^{h}\mathbf{w}^{k+}$. These predictions are combined with image and inverse depth model parameters at the update stage $[\Delta U]$ to create a new estimate $^{k}{}_{\Delta}\mathbf{w}^{k+1}$ and $^{h}\rho^{k+1}$.

Figure 4.6: Structure flow pyramidal filter architecture.

Figure 4.6b illustrates the filter structure for levels $h = 1, \ldots, H-1$. The structure flow output is computed at the reconstruction block $[R]$ using the output of the update block $[\Delta U]$ and the flow from level $h+1$. In the prediction block $[P_{[]}]$, the reconstructed flow is used to create predictions of flow $^{h}\mathbf{w}^{k+}$, image brightness $^{h}Y^{k+}$ and state $\{^{h}{}_{\Delta}\mathbf{w}^{k+}, {}^{h}\rho^{k+}\}$. These predictions are plugged into the update block $[\Delta U]$ and combined with new measurement parameters from $[I_Y]$ and $[I_\rho]$ blocks to compute a new estimate $\{^{h}{}_{\Delta}\mathbf{w}^{k+1}, {}^{h}\rho^{k+1}\}$.

The numerical implementation of the various filter stages are implemented using the spherepix data structure. The reader may refer to Chapter 3, in particular Sections 3.4.1 and 3.5.2 for the details relevant to the filter algorithm.

4.5.1 Brightness parameter extraction

Raw image measurements Y from the camera are used to fit a linear brightness model for each sphere pixel $\boldsymbol{\eta}_{\mathbf{p}}$ where $\mathbf{p} = (i, j)$ is the memory address in the spherepix patch. The model parameters correspond to the constant term $\hat{Y}_{\mathbf{p}} \in \mathbb{R}$ and the image gradient $\partial_{\boldsymbol{\eta}} \hat{Y}_{\mathbf{p}} \in T_{\eta_{\mathbf{p}}} S^2$ at $\mathbf{p}$.

Computation of these parameters is performed in two stages. First, cost function (4.32) is

minimized for parameters $\{\partial_{\boldsymbol{\beta}}\hat{Y}_{\mathbf{p}}, \hat{Y}_{\mathbf{p}}\}$ expressed in local 2D tangent plane coordinates.

$$\varepsilon_{\mathbf{p}} = \sum_{\mathbf{q}\in\Lambda_{\mathbf{p}}} G(\mathbf{p},\mathbf{q}) \| Y_{\mathbf{q}} - \partial_{\boldsymbol{\beta}}\hat{Y}_{\mathbf{p}}(\mathbf{p}-\mathbf{q}) - \hat{Y}_{\mathbf{p}} \|^2 \tag{4.32}$$

A support window $\Lambda_{\mathbf{p}} = 5 \times 5$ pixels centered on $\mathbf{p}$ is used. Gaussian weight mask $G(\mathbf{p},\mathbf{q})$ is generated as $G = g^\top g$ with $g = [1,4,6,4,1]/16$. Thanks to the spatial symmetry of the support window and weight function, it is possible to efficiently solve for $\{\partial_{\boldsymbol{\beta}}\hat{Y}_{\mathbf{p}}, \hat{Y}_{\mathbf{p}}\}$ as a series of 1D convolutions in the row and column axis as illustrated in Figure 4.7. This procedure is the same as the one described in Chapter 2, Section 2.4.1.

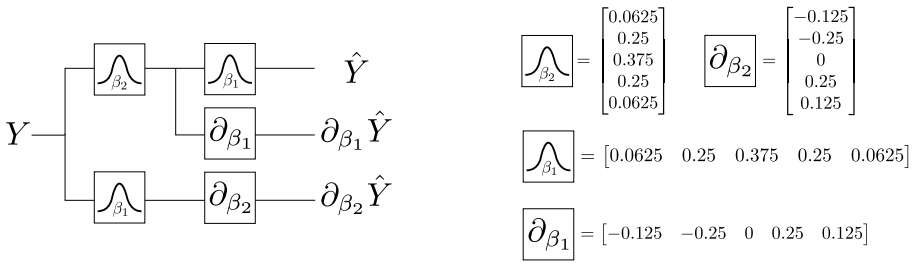


Figure 4.7: Brightness parameters extraction. Gaussian blurring is applied along $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ axis, followed by a derivative filters.

Two-dimensional image gradient $\partial_{\boldsymbol{\beta}}\hat{Y}_{\mathbf{p}}$ is then expressed as a 3D tangent vector $\partial_{\boldsymbol{\eta}}\hat{Y}_{\mathbf{p}}$ following Equation (3.22)

$$\partial_{\boldsymbol{\eta}}\hat{Y}_{\mathbf{p}} = \Delta\mu_{\mathbf{p}}\mathbf{B}_{\mathbf{p}}^\top \partial_{\boldsymbol{\beta}}\hat{Y}_{\mathbf{p}} \tag{4.33}$$

4.5.2 Inverse depth parameter extraction

Provided inverse depth measurements ρ, a linear model is fitted. The model is comprised of parameters $\partial_{\boldsymbol{\eta}}\hat{\rho}_{\mathbf{p}} \in T_{\eta_{\mathbf{p}}}S^2$ for the inverse depth gradient vector and $\hat{\rho}_{\mathbf{p}} \in \mathbb{R}$ for the constant term. In particular for the computation of $\partial_{\boldsymbol{\eta}}\hat{\rho}_{\mathbf{p}}$, one must properly handle object occlusions to remove undesired large gradient vectors. At each pixel location, the gradient vector on both sides of a discontinuity is computed and the appropriate vector is selected. Two-dimensional gradient vector, expressed on Spherepix beta coordinates, $\partial_{\boldsymbol{\beta}}\hat{\rho}_{\mathbf{p}} = (\partial_{\boldsymbol{\beta}_1}\hat{\rho}_{\mathbf{p}}, \partial_{\boldsymbol{\beta}_2}\hat{\rho}_{\mathbf{p}})^\top$ is computed as follows:

$$\partial_{\boldsymbol{\beta}_1}\hat{\rho}_{\mathbf{p}} = \begin{cases} \delta_{\boldsymbol{\beta}_1+}\rho_{\mathbf{p}} & \text{if } |\delta_{\boldsymbol{\beta}_1+}\rho_{\mathbf{p}}| \le |\delta_{\boldsymbol{\beta}_1-}\rho_{\mathbf{p}}| \\ \delta_{\boldsymbol{\beta}_1-}\rho_{\mathbf{p}} & \text{otherwise} \end{cases} \tag{4.34}$$

$$\partial_{\boldsymbol{\beta}_2}\hat{\rho}_{\mathbf{p}} = \begin{cases} \delta_{\boldsymbol{\beta}_2+}\rho_{\mathbf{p}} & \text{if } |\delta_{\boldsymbol{\beta}_2+}\rho_{\mathbf{p}}| \le |\delta_{\boldsymbol{\beta}_2-}\rho_{\mathbf{p}}| \\ \delta_{\boldsymbol{\beta}_2-}\rho_{\mathbf{p}} & \text{otherwise} \end{cases} \tag{4.35}$$

where δ_+, δ_- denote forward and backward difference operators in the $\boldsymbol{\beta}_1$ (column) and $\boldsymbol{\beta}_2$ (row) axis. Table 4.1 defines these as well as the central difference operator.

	$\delta_{\boldsymbol{\beta}_1} u_{ij}$	$\delta_{\boldsymbol{\beta}_2} u_{ij}$
backward	$\delta_{\boldsymbol{\beta}_{1^-}} = u_j - u_{j-1}$	$\delta_{\boldsymbol{\beta}_{2^-}} = u_i - u_{i-1}$
central	$\delta_{\boldsymbol{\beta}_{1^0}} = u_{j+1} - u_{j-1}$	$\delta_{\boldsymbol{\beta}_{2^0}} = u_{i+1} - u_{i-1}$
forward	$\delta_{\boldsymbol{\beta}_{1^+}} = u_{j+1} - u_j$	$\delta_{\boldsymbol{\beta}_{2^+}} = u_{i+1} - u_i$

Table 4.1: Difference operators in beta coordinates.

The 3D vector representation $\partial_{\boldsymbol{\eta}} \hat{\rho}_{\mathbf{p}} \in T_{\eta_p} S^2$ is (Equation (3.22))

$$\partial_{\boldsymbol{\eta}} \hat{\rho}_{\mathbf{p}} = \Delta\mu_{\mathbf{p}} \mathbf{B}_{\mathbf{p}}^{\top} \partial_{\boldsymbol{\eta}} \hat{\rho}_{\mathbf{p}} \tag{4.36}$$

The constant parameter term is given by the raw inverse depth data. That is, $\hat{\rho}_{\mathbf{p}} = \rho_{\mathbf{p}}$.

4.5.3 State prediction ($k \to k+$)

The prediction stage of the filter uses the current state estimate to create predictions of the structure flow forward in time. In the present version of the algorithm, it is assumed that derived from inertial measurements in the PDE (4.27) have negligible. That is,

$$\Omega_{\times} \mathbf{w} - \boldsymbol{a}_{\mathbf{w}} \approx 0. \tag{4.37}$$

In practice, for the high frame rates and the scenarios considered in this chapter, these terms have sub-pixel magnitudes and are at least an order of magnitude less significant than the other terms in the propagation algorithm. In particular, the contribution of this term is proportional to the magnitude of Ω and $\boldsymbol{a}_c$ and inversely proportional to frame rate. Figure 4.8 plots the xyz components of Equation (4.37) for a structure flow field computed from realistic velocities and acceleration values to provide an indication of the validity of this assumption.

For the top level H, the prediction block uses the current coarse estimate of structure flow and the inverse depth field as initial conditions ${}^{\mathrm{H}}\mathbf{w}(0) := {}^{\mathrm{H}}\mathbf{w}^{\mathrm{k}}$ and ${}^{\mathrm{H}}\rho(0) := {}^{\mathrm{H}}\rho^{\mathrm{k}}$ for the PDEs (4.27) and (4.29) with assumption (4.37). These PDEs are solved numerically to create the state predictions ${}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+}} := {}^{\mathrm{H}}\mathbf{w}(1)$ and ${}^{\mathrm{H}}\rho^{\mathrm{k+}} := {}^{\mathrm{H}}\rho(1)$ for the next frame $k+1$. The explicit PDEs that are solved at level H are

$$\frac{\partial {}^{\mathrm{H}}\mathbf{w}}{\partial t} = -\frac{\partial {}^{\mathrm{H}}\mathbf{w}}{\partial \boldsymbol{\eta}} \mathbf{P}_{\boldsymbol{\eta}} {}^{\mathrm{H}}\mathbf{w} - {}^{\mathrm{H}}\mathbf{w} \langle \boldsymbol{\eta}, {}^{\mathrm{H}}\mathbf{w} \rangle \tag{4.38}$$

$$\frac{\partial {}^{\mathrm{H}}\rho}{\partial t} = -\frac{\partial {}^{\mathrm{H}}\rho}{\partial \boldsymbol{\eta}} \mathbf{P}_{\boldsymbol{\eta}} {}^{\mathrm{H}}\mathbf{w} - {}^{\mathrm{H}}\rho \langle \boldsymbol{\eta}, {}^{\mathrm{H}}\mathbf{w} \rangle \tag{4.39}$$

For lower levels $h = 1, \ldots, H-1$, the propagation scheme is defined by a PDE system modeling the transport of the structure flow, image brightness and state $\{{}^{\mathrm{h}}\Delta\mathbf{w}, {}^{\mathrm{h}}\rho\}$ by the

reconstructed structure flow ${}^{\mathrm{h}}\mathbf{w}$ at each level.

$$\frac{\partial {}^{\mathrm{h}}\mathbf{w}}{\partial t} = -\frac{\partial {}^{\mathrm{h}}\mathbf{w}}{\partial \boldsymbol{\eta}} \mathbf{P}_{\eta} {}^{\mathrm{h}}\mathbf{w} - {}^{\mathrm{h}}\mathbf{w} \langle \boldsymbol{\eta}, {}^{\mathrm{h}}\mathbf{w} \rangle \tag{4.40}$$

$$\frac{\partial {}^{\mathrm{h}}{}_{\Delta}\mathbf{w}}{\partial t} = -\frac{\partial {}^{\mathrm{h}}{}_{\Delta}\mathbf{w}}{\partial \boldsymbol{\eta}} \mathbf{P}_{\eta} {}^{\mathrm{h}}\mathbf{w} - {}^{\mathrm{h}}{}_{\Delta}\mathbf{w} \langle \boldsymbol{\eta}, {}^{\mathrm{h}}\mathbf{w} \rangle \tag{4.41}$$

$$\frac{\partial {}^{\mathrm{h}}\rho}{\partial t} = -\frac{\partial {}^{\mathrm{h}}\rho}{\partial \boldsymbol{\eta}} \mathbf{P}_{\eta} {}^{\mathrm{h}}\mathbf{w} - {}^{\mathrm{h}}\rho \langle \boldsymbol{\eta}, {}^{\mathrm{h}}\mathbf{w} \rangle \tag{4.42}$$

$$\frac{\partial {}^{\mathrm{h}}Y}{\partial t} = -\frac{\partial {}^{\mathrm{h}}Y}{\partial \boldsymbol{\eta}} \mathbf{P}_{\eta} {}^{\mathrm{h}}\mathbf{w} \tag{4.43}$$

Initial conditions are set to ${}^{\mathrm{h}}\mathbf{w}(0) := {}^{\mathrm{h}}\mathbf{w}^{\mathrm{k}}$, ${}^{\mathrm{h}}{}_{\Delta}\mathbf{w}(0) := {}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k}}$, ${}^{\mathrm{h}}\rho(0) := {}^{\mathrm{h}}\rho^{\mathrm{k}}$ and ${}^{\mathrm{h}}Y(0) := {}^{\mathrm{h}}Y^{\mathrm{k}}$ and the system is run for one time step unit. Details on the numerical solution to these equations are provided in Section 4.6.

4.5.4 State update ($k+ \rightarrow k+1$)

In the update stage of the filter, predictions made for time $k+1$ are corrected using new image and inverse depth measurements. There are two update schemes used: one for top level H to create the updated state ${}^{\mathrm{H}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}}$ and one applied to the remaining lower levels to update ${}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}}$. The update of the inverse depth state ${}^{\mathrm{h}}\rho^{\mathrm{k+1}}$ follows the same scheme for all pyramid levels.

For level H, the update stage creates a new estimate ${}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}}$ using low resolution data available at level H. For level H, cost function (4.44) uses low resolution image and inverse depth measurements, as well as the predicted flow from the propagation stage to create a new structure flow estimate ${}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}}$. The cost function

$${}^{H}\varepsilon_{\mathbf{w}} = \gamma_1 \| {}^{H}E_Y \|^2 + \gamma_2 \| {}^{H}E_{\rho} \|^2 + \gamma_3 \| {}^{H}E_t \|^2 \tag{4.44}$$

consists of three data terms ${}^{H}E_Y$, ${}^{H}E_{\rho}$ and ${}^{H}E_t$ using image, inverse depth and the predicted flow, respectively. Scalar gains γ_1, γ_2, γ_3 control the relative weight of each term.

The image data term E_Y is based on the image conservation PDE (4.25). It is defined as

$${}^{H}E_Y = \partial_{\boldsymbol{\eta}} \hat{Y}^{k+1} \mathbf{P}_{\eta} {}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}} + \Delta\mu^2 (\hat{Y}^{k+1} - \hat{Y}^{k}) \tag{4.45}$$

where $\partial_{\boldsymbol{\eta}} \hat{Y}^{k+1}$, $\hat{Y}^{k+1}$ and $\hat{Y}^{k}$ are the image gradient and constant terms computed at the image model block. The temporal difference term is multiplied by $\Delta\mu^2$ to compensate for the transformation of gradient vector $\partial_{\boldsymbol{\eta}} \hat{Y}^{k+1}$ to tangent space coordinates (Eq. (3.22)) and the multiplication by $\mathbf{P}_{\eta} {}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}}$. Using this term, it is only possible to recover the component of $\mathbf{w}$ lying on the tangent space of $\boldsymbol{\eta}$, that is, the optical flow $\boldsymbol{\Phi}(\boldsymbol{\eta}, t)$ subject to the aperture problem.

The inverse depth data term E_{ρ} uses the inverse depth conservation PDE (4.29) to recover the structure flow from state ρ^k and measurements and $\hat{\rho}^{k+1}$. The data term is

$${}^{H}E_{\rho} = \partial_{\boldsymbol{\eta}} \hat{\rho}^{k+1} \mathbf{P}_{\eta} {}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}} + \Delta\mu^2 \left((\hat{\rho}^{k+1} - \rho^{k}) + \hat{\rho}^{k+1} \langle \boldsymbol{\eta}, {}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}} \rangle \right) \tag{4.46}$$

Analogously to the image cost term, the scalar component of Equation (4.46) is multiplied by

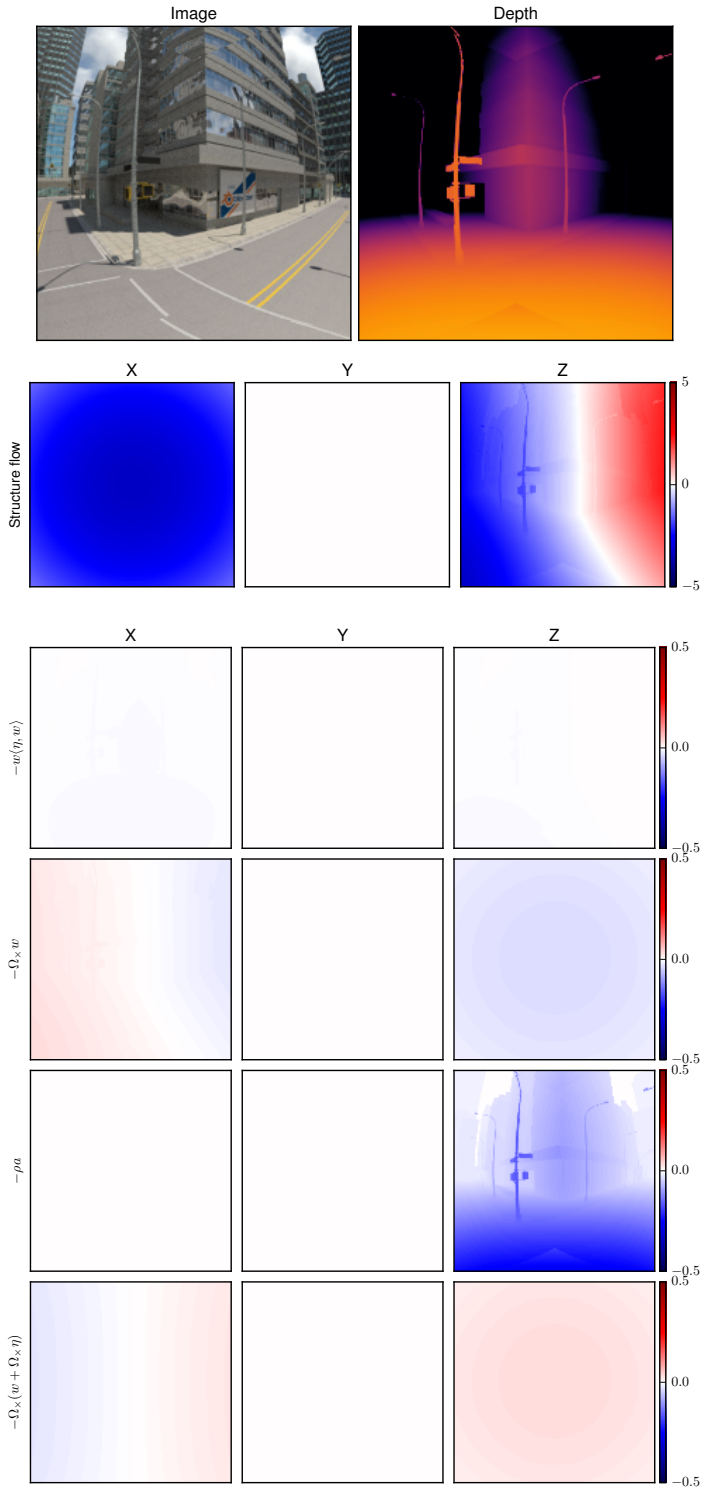


Figure 4.8: Contribution of each source term in PDE (4.27). The simulated camera works at 300 Hz and moves with a linear velocity of $5^m/_s$ and acceleration of $1^m/_{s^2}$ both in the z direction. Angular velocity is set to $180\,^{\text{deg}}/_s$. The maximum contribution is provided by the $-\rho \boldsymbol{a}_c$ term, but is still less than 0.2 pixels.

$\Delta\mu^2$ to compensate for the transformation of vector quantities to 3D tangent space coordinates. As PDE (4.29) includes the effect of $\mathbf{w}$ along normal direction $\boldsymbol{\eta}$, it is possible to recover both tangent and normal components of the flow field. For the tangent component, it still suffers from a form of aperture problem where only flow in the direction of the inverse depth gradient can be recovered Spies et al. [2002].

Finally, the temporal smoothing term E_t includes the predicted flow $\mathbf{w}^{\mathrm{H}}_{k+}$ as prior solution to the cost function

$$^{H}E_t = {}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}} - {}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+}} \tag{4.47}$$

This regularization term guarantees there is a solution for the structure flow at each pixel. In regions where it is not possible to recover flow from the data, the predicted flow ${}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+}}$ will act as best estimate for time $k+1$.

Cost function (4.44) describes a linear least squares problem with respect to ${}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}}$. The solution can be expressed as a linear system of the form:

$$A\,{}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}} = b \tag{4.48}$$

This system can efficiently be solved in parallel for each pixel using the Cholesky decomposition of A and forward/backward substitution. After solving Equation (4.48), an average smoothing filter of size 5×5 is applied to the resulting flow field ${}^{\mathrm{H}}\mathbf{w}^{\mathrm{k+1}}$ in order to accelerate the diffusion of flow estimates to textureless regions of the image.

For levels $h = 0, \ldots, H-1$, the update stage computes new state estimates ${}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}}$ provided with the predicted state and new measurement data. Cost function (4.49) is defined as

$$^{h}\varepsilon_{\mathbf{w}} = \gamma_1 \|{}^{h}E_Y\|^2 + \gamma_2 \|{}^{h}E_\rho\|^2 + \gamma_3 \|{}^{h}E_t\|^2 \tag{4.49}$$

where the data terms for image, inverse depth and temporal smoothing defined are

$$^{h}E_Y = \partial_{\boldsymbol{\eta}}\hat{Y}^{k+1}\mathbf{P}_{\boldsymbol{\eta}}\,{}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}} + \Delta\mu^2(\hat{Y}^{k+1} - \hat{Y}^{k+}) \tag{4.50}$$

$$^{h}E_\rho = \partial_{\boldsymbol{\eta}}\hat{\rho}^{k+1}\mathbf{P}_{\boldsymbol{\eta}}\,{}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}} + \Delta\mu^2[(\hat{\rho}^{k+1} - \rho^{k+}) + \hat{\rho}^{k+1}\langle\boldsymbol{\eta}, {}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}}\rangle] \tag{4.51}$$

$$^{h}E_t = {}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}} - {}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+}} \tag{4.52}$$

Equation (4.49) describes a linear least squares problem with respect to unknown ${}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}}$, and can be solved following the same procedure used for top level H. Once ${}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}}$ has been computed, the structure flow at level h is reconstructed using Equation (4.31). That is

$${}^{\mathrm{h}}\mathbf{w}^{\mathrm{k+1}} = {}^{\mathrm{h+1:h}}\mathbf{w}^{\mathrm{k+1}} + {}^{\mathrm{h}}{}_{\Delta}\mathbf{w}^{\mathrm{k+1}} \tag{4.53}$$

The flow ${}^{\mathrm{h}}\mathbf{w}^{\mathrm{k+1}}$ is smoothed using an average filter for a given number of iterations. Once completed, the resulting flow is cascaded to the next level below until level $h = 1$ is reached and the flow at the original resolution is computed.

The formulation of the inverse depth state update is equal for all pyramid levels. Cost function (4.54) considers both current measurements and the predicted inverse depth from

previous frame for estimating new state ${}^{\mathrm{h}}\rho^{\mathrm{k+1}}$. The cost function is

$$ {}^{h}\varepsilon_{\rho} = \gamma_4 \| {}^{\mathrm{h}}\rho^{\mathrm{k+1}} - {}^{\mathrm{h}}\hat{\rho}^{\mathrm{k+1}} \|^2 + \gamma_5 \| {}^{\mathrm{h}}\rho^{\mathrm{k+1}} - {}^{\mathrm{h}}\rho^{\mathrm{k+}} \|^2 \tag{4.54} $$

where γ_4 and γ_5 are the relative weights between measurements and prediction. The least squares solution pf ${}^{\mathrm{h}}\rho^{\mathrm{k+1}}$ is

$$ {}^{\mathrm{h}}\rho^{\mathrm{k+1}} = \frac{\gamma_4 {}^{\mathrm{h}}\hat{\rho}^{\mathrm{k+1}} + \gamma_5 {}^{\mathrm{h}}\rho^{\mathrm{k+}}}{\gamma_4 + \gamma_5} \tag{4.55} $$

In places where no measurements are available, $\gamma_4 = 0$ and the updated inverse depth is ${}^{\mathrm{h}}\rho^{\mathrm{k+}} = {}^{\mathrm{h}}\rho^{\mathrm{k+1}}$.

4.6 Numerical prediction

Considering the real-time nature of the structure flow filtering algorithm, one desires a numerical scheme that is fast and robust to noisy input for solving the propagation equations in Section 4.5.3. A numerical scheme based on upwind finite differences Thomas [1995] is designed following a similar approach as that developed for the optical flow filter algorithm in Chapter 2.

Consider Equation (4.56) as a discrete version of prediction Equations (4.38) and (4.40).

$$ \frac{\mathbf{w}^{n+1} - \mathbf{w}^{n}}{\Delta t} = -\frac{\partial \mathbf{w}^{n}}{\partial \boldsymbol{\eta}} \mathbf{P}_{\boldsymbol{\eta}} \mathbf{w}^{n} - \mathbf{w}^{n} \langle \boldsymbol{\eta}, \mathbf{w}^{n} \rangle \tag{4.56} $$

For convenience in the notation, the pyramid level index h and pixel coordinate (i, j) are omitted, as the scheme runs identically at every level and for each pixel. Index n refers to an internal time index for the numerical scheme running for $n = 0, \ldots, N-1$ with $\Delta t = {}^{1}/_{N}$ and N being a parameter describing the number of iterations. Initial conditions of the problem are set to $\mathbf{w}^0 := \mathbf{w}^k$ and the output of the scheme is $\mathbf{w}^{N-1} =: \mathbf{w}^{k+}$.

Similar discrete versions of Equations (4.39), (4.41) and (4.43) can be derived. In this case, the discrete equations are linear PDEs with respect to $\mathbf{w}$ and can be solved using the same numerical scheme used for Equation (4.56).

The unknown variable $\mathbf{w}^{n+1}$ can be calculated as the sum of the previous state $\mathbf{w}^n$ plus the contributions of the structure flow propagated in tangent space by the optical flow plus the source term. Equation (4.56) is a non-linear PDE on $\mathbf{w}$ for which direct application of finite difference methods do not yield the expected results [Thomas, 1999, p. 140]. To break this non-linearity in a way that allows fast computations, the idea of *dominant optical flow* (Chapter 2.5.2) is used. The dominant optical flow is the largest optical flow vector in the 4-neighborhood of a pixel.

The optical flow in 2D tangent plane coordinates and pixel units is computed as

$$ \Phi_{ij} := \begin{pmatrix} u_{ij} \\ v_{ij} \end{pmatrix} = \frac{1}{\Delta \mu} \mathbf{B}_{\boldsymbol{\eta}_{ij}} \mathbf{P}_{\boldsymbol{\eta}_{ij}} \mathbf{w}_{ij} \tag{4.57} $$

The dominant optical flow $\hat{\Phi}_{ij} = (\hat{u}^n_{ij}, \hat{v}^n_{ij})$ is calculated as the largest flow component in the

row and column axis

$$\hat{u}_{ij} = \begin{cases} u_{i,j-1} & \text{if } \delta_{\boldsymbol{\beta}_{1^0}}|u_{ij}| > 0 \\ u_{i,j+1} & \text{otherwise} \end{cases} \tag{4.58}$$

$$\hat{v}_{ij} = \begin{cases} v_{i-1,j} & \text{if } \delta_{\boldsymbol{\beta}_{2^0}}|v_{ij}| > 0 \\ v_{i+1,j} & \text{otherwise} \end{cases} \tag{4.59}$$

where $\delta_{\boldsymbol{\beta}_{1^0}}$ and $\delta_{\boldsymbol{\beta}_{2^0}}$ are central difference operators defined in Table 4.1.

Functions $\delta_{\boldsymbol{\beta}_1}(f;\hat{u})$ and $\delta_{\boldsymbol{\beta}_2}(f;\hat{v})$ define the upwind finite difference operators in column and row axis for a given field f. They are computed as

$$\delta_{\boldsymbol{\beta}_1}(f;\hat{u}) = \begin{cases} \delta_{\boldsymbol{\beta}_{1^-}} f & \text{if } \hat{u} > 0 \\ \delta_{\boldsymbol{\beta}_{1^+}} f & \text{otherwise} \end{cases} \tag{4.60}$$

$$\delta_{\boldsymbol{\beta}_2}(f;\hat{v}) = \begin{cases} \delta_{\boldsymbol{\beta}_{2^-}} f & \text{if } \hat{v} > 0 \\ \delta_{\boldsymbol{\beta}_{2^+}} f & \text{otherwise} \end{cases} \tag{4.61}$$

These operators evaluate the direction of the dominant flow in the column and row axis and select the adequate upwind difference operator.

The iterations of the numerical scheme are as follows. First, dominant optical flow $\hat{u}^n$ is calculated from $\mathbf{w}^n$. Then, the propagation takes place in the $\boldsymbol{\beta}_1$ (column) axis for all points in the spherical grid as follows

$$\mathbf{w}^* = \mathbf{w}^n - \Delta t \left[\hat{u}^n \delta_{\boldsymbol{\beta}_1}(\mathbf{w}^n;\hat{u}^n) + \mathbf{w}^n \langle \boldsymbol{\eta}, \mathbf{w}^n \rangle\right] \tag{4.62}$$

$$_{\vartriangle}\mathbf{w}^* = {}_{\vartriangle}\mathbf{w}^n - \Delta t \left[\hat{u}^n \delta_{\boldsymbol{\beta}_1}({}_{\vartriangle}\mathbf{w}^n;\hat{u}^n) + {}_{\vartriangle}\mathbf{w}^n \langle \boldsymbol{\eta}, \mathbf{w}^n \rangle\right] \tag{4.63}$$

$$\rho^* = \rho^n - \Delta t \left[\hat{u}^n \delta_{\boldsymbol{\beta}_1}(\rho^n;\hat{u}^n) + \rho^n \langle \boldsymbol{\eta}, \mathbf{w}^n \rangle\right] \tag{4.64}$$

$$Y^* = Y^n - \Delta t \hat{u}^n \delta_{\boldsymbol{\beta}_1}(Y^n;\hat{u}^n) \tag{4.65}$$

producing the propagated fields $\mathbf{w}^*$, $_{\vartriangle}\mathbf{w}^*$, ρ^* and Y^*. Next, the dominant flow $\hat{v}^*$ is calculated from $\mathbf{w}^*$ and the propagation takes place in the $\boldsymbol{\beta}_2$ (row) axis

$$\mathbf{w}^{n+1} = \mathbf{w}^* - \Delta t \left[\hat{v}^* \delta_{\boldsymbol{\beta}_2}(\mathbf{w}^*;\hat{v}^*) + \mathbf{w}^* \langle \boldsymbol{\eta}, \mathbf{w}^* \rangle\right] \tag{4.66}$$

$$_{\vartriangle}\mathbf{w}^{n+1} = {}_{\vartriangle}\mathbf{w}^* - \Delta t \left[\hat{v}^* \delta_{\boldsymbol{\beta}_2}({}_{\vartriangle}\mathbf{w}^*;\hat{v}^*) + {}_{\vartriangle}\mathbf{w}^* \langle \boldsymbol{\eta}, \mathbf{w}^* \rangle\right] \tag{4.67}$$

$$\rho^{n+1} = \rho^* - \Delta t \left[\hat{v}^* \delta_{\boldsymbol{\beta}_1}(\rho^*;\hat{u}^*) + \rho^* \langle \boldsymbol{\eta}, \mathbf{w}^* \rangle\right] \tag{4.68}$$

$$Y^{n+1} = Y^* - \Delta t \hat{v}^* \delta_{\boldsymbol{\beta}_2}(Y^*;\hat{v}^*) \tag{4.69}$$

The numerical scheme is stable if inequality in Equation (4.70) is satisfied for all points in the grid for all n.

$$\Delta t \max_{n,i,j} \left\{|\hat{u}^n_{ij}|, |\hat{v}^n_{ij}|\right\} \leq 1 \tag{4.70}$$

Given this stability condition where $\Delta t = {}^1/_N$, one needs to choose a suitable number of iterations N of the numerical scheme depending on the maximum flow expected in the application. An important remark is that, as the camera frame rate increases, the magnitude of the estimated flow decreases and hence, less numerical iterations are required.

Figure 4.9 illustrates the results of applying the numerical propagation scheme to the

Bunny sequence during 50 frames. Similar to the optical flow propagation results in Chapter 2.5, the numerical scheme for propagating structure flow has the same numerical artifacts at the *leading* and *trailing* edges. The sharpness, that is, the shock wave, at the leading edge is preserved along time as the foreground object is occluding the background. In contrast, the trailing edge of the foreground objects (e.g. right side of the bunny) blends the brightness, inverse depth and structure flow between the foreground and the background.

4.7 Experimental Results

4.7.1 Ground truth evaluation

To validate the accuracy of the proposed filter algorithm, it is required video sequences recorded at high frame rates that sample the dynamics of the environment smoothly. The Blender source files of the Urban Canyon Dataset from Zhang *et. al.* Zhang et al. [2016] are used to render a photo-realistic city environment in which a 300Hz camera is attached to the front part of a moving vehicle. For the evaluation, only a subset of the trajectory composed of 10000 frames is considered (Figure 4.10b). Images are rendered using a 360 degrees panoramic camera and then mapped to a front facing Spherepix grid of 512×512 pixels. The structure flow ground truth $\mathbf{w}_{\text{gt}}$ is computed for each pixel from the rendered depth map and the camera linear and angular velocity using Equation (4.14) with $\mathbf{v}_{\mathbf{x}} \equiv 0$.

The *Root Mean Square Error* (RMSE) and *Absolute Angular Error* (AAE) are used to measure the accuracy of the algorithm relative to ground truth. The RMSE is calculated by

$$\text{RMSE} = \left\| \frac{\mathbf{w}_{\text{gt}} - \mathbf{w}}{\Delta\mu} \right\| \tag{4.71}$$

and the AAE is computed as

$$\text{AAE} = \arccos\left(\frac{1 + \langle \mathbf{w}_{\text{gt}}, \mathbf{w} \rangle}{\sqrt{1 + \mathbf{w}_{\text{gt}}} \sqrt{1 + \mathbf{w}}} \right) \tag{4.72}$$

For displaying purposes, the 3D structure flow $\mathbf{w}$ is split into tangent and normal components. Tangent flow, that is, the optical flow defined in Equation (4.19), is transformed to 2D plane coordinates and is expressed in pixel units following Equation (4.73). It is possible to use the standard optical flow color wheel encoding Baker et al. [2011] to represent the tangent flow vector field as a color image. The normal flow, Equation (4.74), is the component of the structure flow along direction $\boldsymbol{\eta}$. It is measured in pixel units and plotted as a scalar field.

$$\mathbf{w}_{\perp} = \frac{\mathbf{B}_{\eta}^{\top} \mathbf{P}_{\eta} \mathbf{w}}{\Delta\mu} \tag{4.73}$$

$$\mathbf{w}_{\parallel} = \frac{\langle \boldsymbol{\eta}, \mathbf{w} \rangle}{\Delta\mu} \tag{4.74}$$

Figure 4.11 shows a selection of the first 300 images corresponding to 1 second of real-life time. The initial condition of the structure flow is set to zero. The flow is first identified at

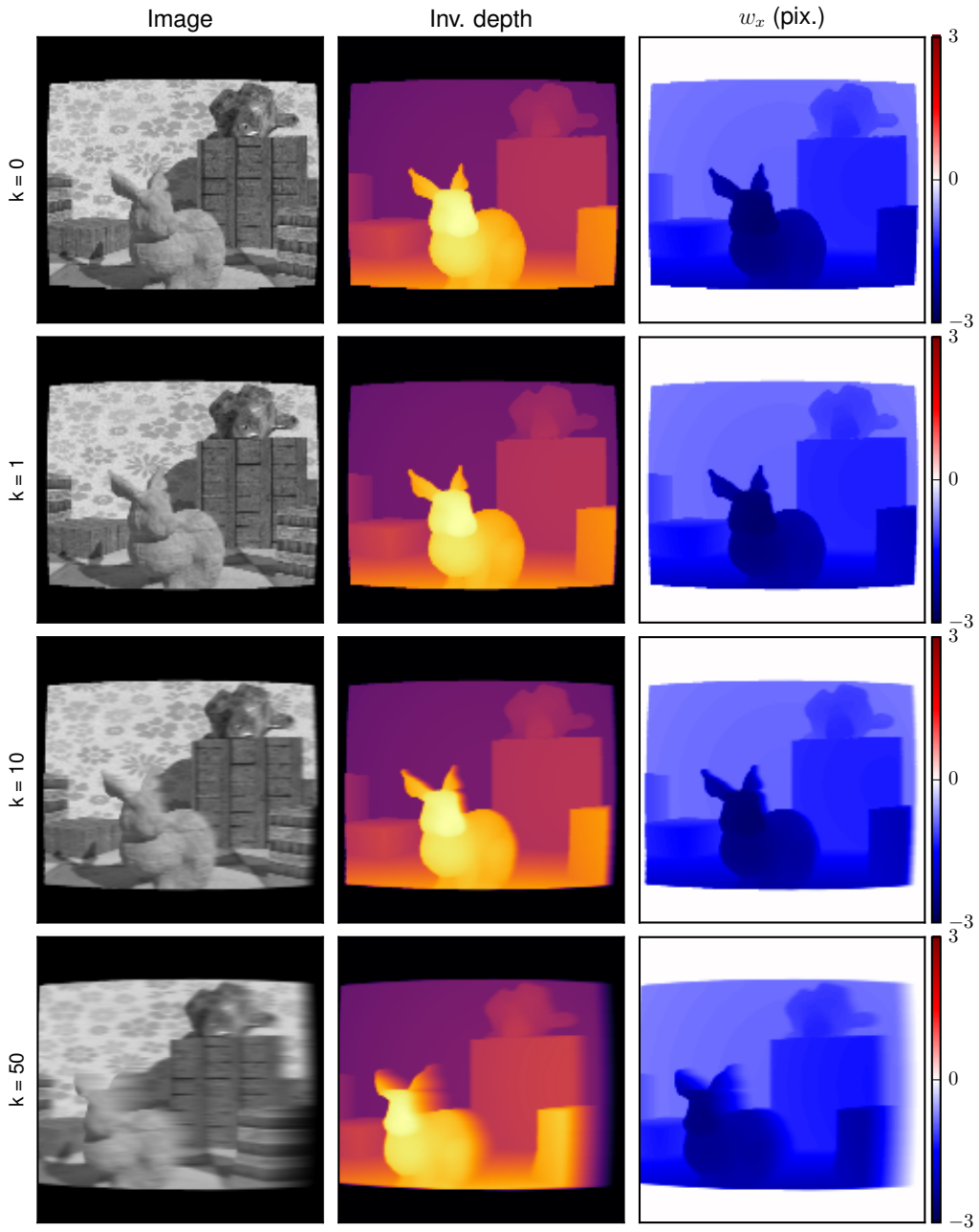
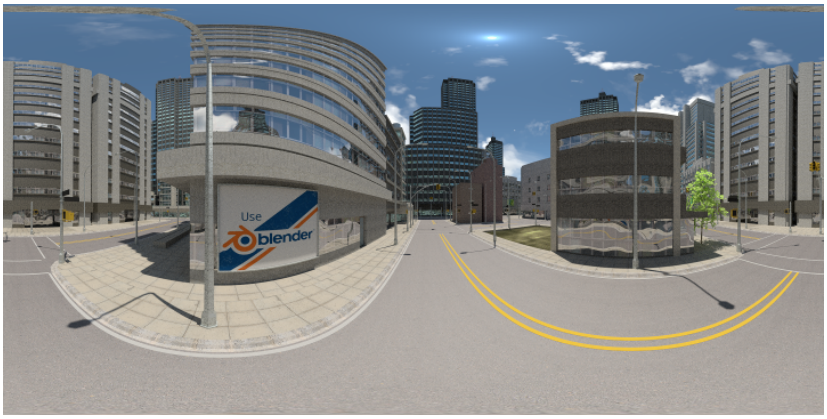
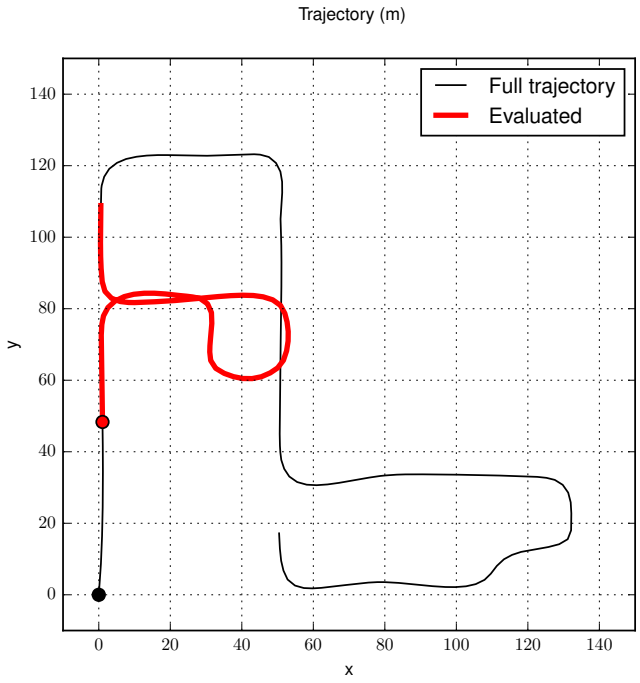


Figure 4.9: Open-loop numerical propagation of image, inverse depth and structure flow for 50 frames. The initial conditions are computed from the rendered depth of the scene and the camera velocity (linear velocity in x). The number of numerical iterations between frame steps is set to $N = 4$.

regions with brightness or depth discontinuities and it is diffused to textureless regions. Over time, a dense flow field is estimated and maintained using new measurements. Convergence of the algorithm can be visualized through the RMSE field plot (bottom row). Regions with



(a) Panoramic rendering.



(b) Vehicle trajectory.

Figure 4.10: Urban Canyon dataset.

higher error are localized at occlusion boundaries, where the algorithm needs to re-identify flow.

Figure 4.12a plots the average RMSE error for the evaluated sequence. After approximately 150 frames, the filter algorithm has converged to a dense flow estimate. After this, the RMSE remains stable around 0.3 pixels and increases in parts of the sequence with high camera rotation, as the algorithm needs to adapt to the changes in velocity. Figure 4.12b displays the average AAE, which stays approximately constant at 20 degrees.

In terms of runtime performance, Tables 4.2 and 4.3 shows the estimated runtime and frame rates of the algorithm configured with one and two pyramid levels respectively. For

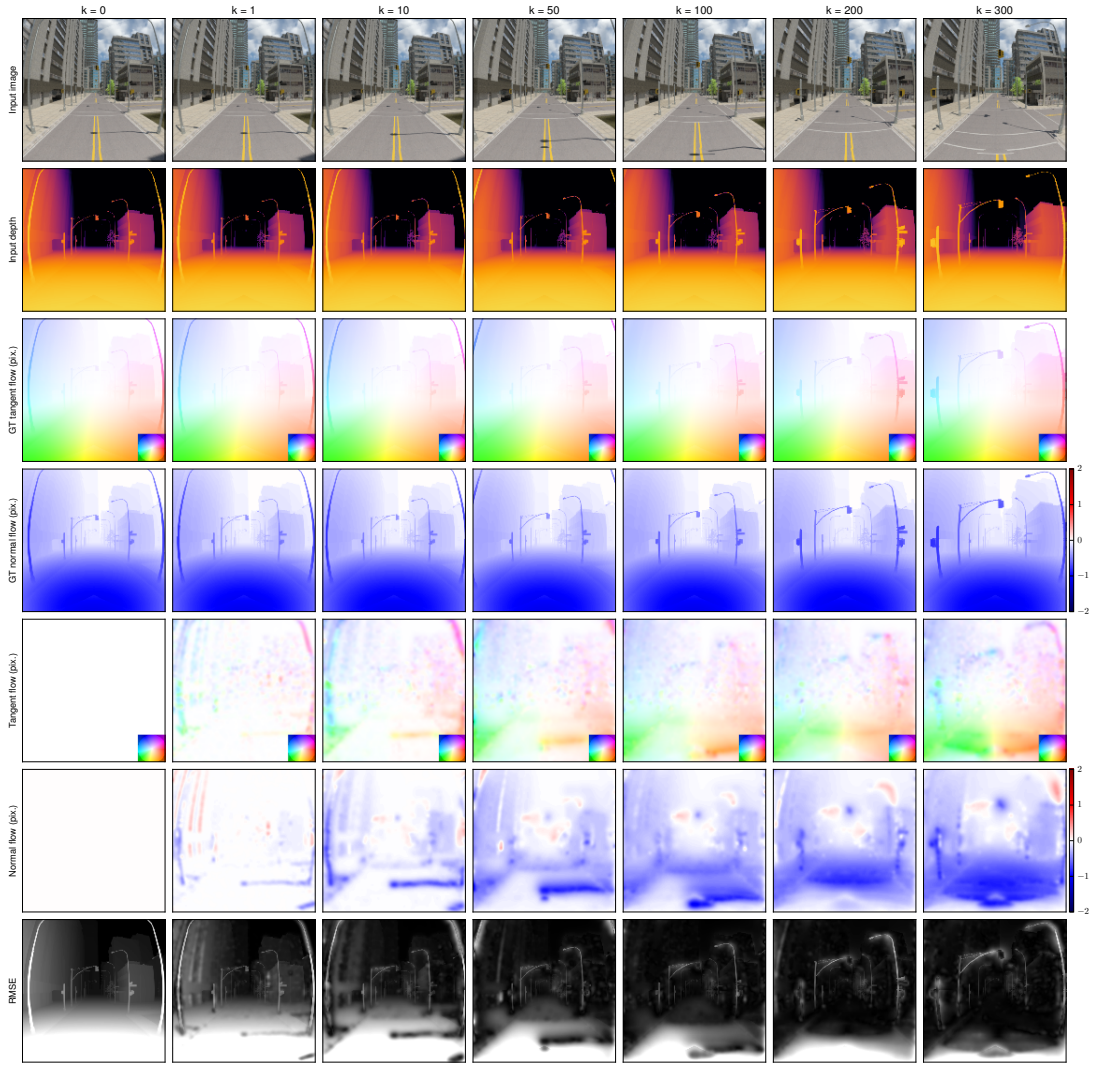
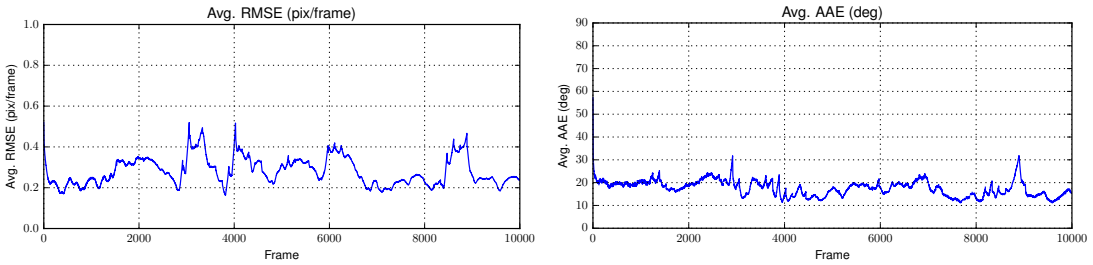


Figure 4.11: Results on the Urban Canyon dataset. The sequence plots the estimated structure flow and ground truth for the first second (300 frames) of video. Structure flow is identified at brightness or depth discontinuities and is diffused to textures regions of the image.

each configuration, the maximum flow allowed in the algorithm is changed, which affects the number of numerical iterations required in the propagation stage. Runtime is considered as the sum of time required to map image and depth measurements onto the Spherepix image plus the runtime of the flow filter itself. Memory transfer operations between CPU and GPU memory spaces are excluded, as they can be overlaid with the execution of GPU kernel code.

On average, the algorithm runs approximately at 600 Hz with 8 pixels maximum flow. Although the number of operations per pixel is deterministic given parameters such as smooth iterations and maximum flow, there is some variability in the reported runtime, which can be attributed to GPU task swapping CPU process scheduling.



(a) Average Root Mean Square Error.

(b) Average Absolute Angular Error.

Figure 4.12: Error metrics for the Urban Canyon dataset.

Max flow (pixels)	Runtime (ms)	Frequency (Hz)	Throughput (Mpix/s)
1	0.929 +/- 0.3	1133.9 +/- 176.9	283.483 +/- 44.2
2	1.029 +/- 0.2	998.1 +/- 111.5	249.521 +/- 27.9
4	1.341 +/- 0.2	755.6 +/- 67.1	188.896 +/- 16.8

Table 4.2: Runtime performance with one pyramid level at 512×512 resolution. Smooth iterations set to 2.

Max flow (pixels)	Runtime (ms)	Frequency (Hz)	Throughput (Mpix/s)
1	1.355 +/- 0.2	747.4 +/- 65.5	186.838 +/- 16.4
2	1.405 +/- 0.2	720.5 +/- 63.4	180.126 +/- 15.9
4	1.501 +/- 0.2	674.3 +/- 59.0	168.584 +/- 14.7
8	1.700 +/- 0.2	594.7 +/- 51.2	148.680 +/- 12.8

Table 4.3: Runtime performance with two pyramid levels at 512×512 resolution. Smooth iterations set to [2, 4] for bottom and top level respectively.

4.7.2 Evaluation on real-life data

A set of experiments using real life data captured using a ZED Stereo Camera were performed to validate the algorithm on real-life image sequences. Stereo video is captured at a resolution of 1440×720 per camera at 60 Hz. The manufacturer's software is used to extract depth from the stereo video and use it in our structure flow algorithm. In this case, a Spherepix patch of 1024×1024 resolution is used. This resolution approximately matches the resolution and field of view of the input camera. The algorithm is configured with two pyramid levels and maximum flow of 8 pixels. Figure 4.13 displays the results for some selected scenes in driving scenarios with moving vehicles.

An important difference of the estimated flow compared to conventional optical flow is the ability to directly distinguish objects getting closer or farther from the camera, as can be observed in the normal flow plots (column 4). In case there is no depth data available, the normal component of the structure flow is poorly observed and the proposed structure flow algorithm generates an estimate that is closer to the optical flow. Table 4.4 shows the

average runtime for this configuration varying the maximum flow. On average, our method achieves frame rates of 172 Hz, which is easily sufficient for use in control loops for even highly dynamic mobile robotic vehicles.

Max flow (pixels)	Runtime (ms)	Frequency (Hz)	Throughput (Mpix/s)
1	4.683 +/- 0.2	213.889 +/- 8.2	213.889 +/- 8.2
2	4.826 +/- 0.2	207.480 +/- 6.9	207.480 +/- 6.9
4	5.154 +/- 0.2	194.258 +/- 6.3	194.258 +/- 6.3
8	5.815 +/- 0.2	172.149 +/- 5.1	172.149 +/- 5.1

Table 4.4: Runtime performance with two pyramid levels at 1024×1024 resolution. Smooth iterations set to [2, 4] for bottom and top level respectively.

4.8 Summary

This chapter introduced the structure flow field as the three-dimensional vector field encoding the velocity of the surrounding environment relative to the camera body fixed frame and scaled by the inverse depth. Partial differential equations on spherical geometry were developed to model the temporal evolution of the structure flow field, image brightness, depth and inverse depth fields. These PDEs can be used both to derive predictors to propagate these quantities forward in time, or as innovation terms to estimate the structure flow from measured data.

A real-time algorithm for computing structure flow is proposed. The algorithm is based on a filtering architecture that incrementally computes the flow from streams of brightness and depth data. The algorithm achieves frame rates of the order of 600 Hz at 512×512 pixel resolution and 172 Hz at 1024×1024 for flow vectors up to 8 pixels. Experimental validation of the algorithm using simulated high-speed video with ground truth is provided, as well as results on real-life video sequences using a stereo camera.

Assumption (4.37) is a practical assumption that allows implementation of the algorithm in the absence of a calibrated and time-synchronized inertial measurement unit. Such a system was unavailable for the field tests and involves considerable complexity in systems integration given that the stereo cameras used have never been designed to be integrated with a real-time IMU system. It is clear that inclusion of these terms will improve results, although it is not believed that the improvement will be particularly significant compared to the results obtained. More significantly, the current formulation of the filter algorithm relies on the availability of depth measurements. A natural extension is to express the cost functions relying on depth measurements in terms of image disparity for a coupled stereo pair. Such approach will allow us to formulate the state update equations purely on image data.

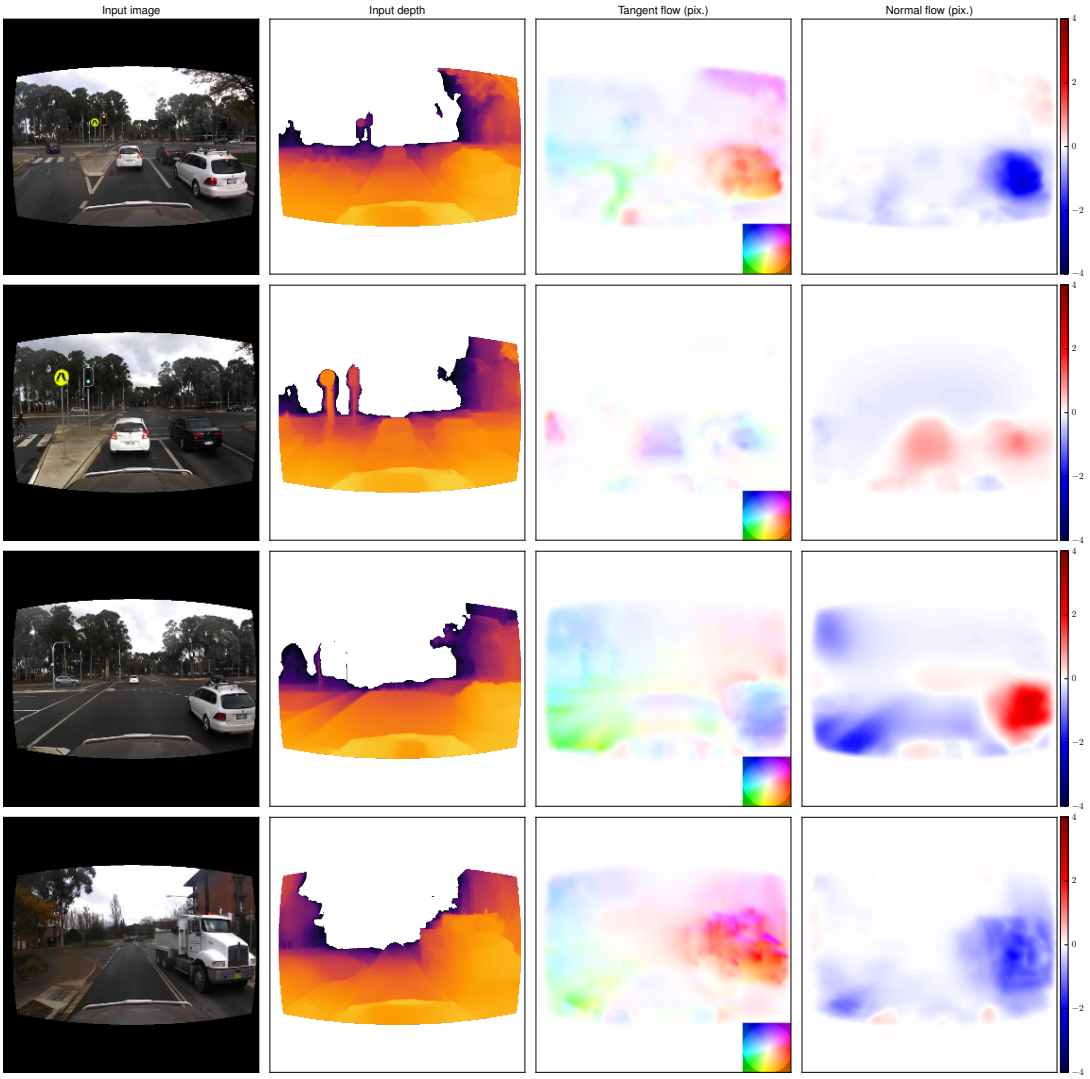


Figure 4.13: Results on real-life stereo video sequences. The camera captures 1440×720 images at 60 Hz and computes depth. The brightness and estimated depth are used in the filter algorithm to compute the underlying structure flow. The normal flow component of the structure flow offers immediate information about objects getting close to or far from the camera (rows 1 and 3).

Chapter 5

Conclusions

This thesis introduced new real-time visual flow algorithms for robotic applications. The first example of such algorithms is an optical flow filter capable of running up to 800 Hz at 640×480 image resolution. An important aspect of the algorithm is that of developing predictors for image and optical flow to propagate current state estimates forward in time. This is achieved by modeling the evolution of the image and optical flow as systems of partial differential equations which is then solved numerically on GPU.

Moreover, a new type of visual flow named *structure flow* is introduced. Intuitively, structure flow is the three-dimensional vector field describing the velocity of the environment relative to the camera as seen by the camera. Geometrically, it is the relative 3D velocity of the environment sampled at each pixel and divided by the inverse distance, and it is measured in radians per second. Structure flow is a generalization of optical flow as it contains information about the motion in the normal direction of the camera. Partial differential equations are used to model the spatio-temporal evolution of the structure flow and associated image brightness and depth fields. These equations can be used to create predictors to propagate such quantities forward in time, as well as to use them to estimate the underlying flow from image and depth measurements.

To implement the structure flow algorithm, the *Spherepix* was developed to efficiently represent data on the sphere. Pixels on Spherepix images are arranged such that locally they approximately satisfy the properties of equidistance and orthogonality. These properties are fundamental to achieve efficient implementation of low-level image processing routines such as blurring and gradient computation. Other algorithms such as SIFT feature point extraction and optical flow can be efficiently implemented on spherepix.

5.1 Achievements

The following is a summary of the achievements of this thesis:

- **Optical flow filter:** An open-source filtering algorithm for the computation of optical flow has been formulated and developed. The algorithm is formulated as update-prediction blocks. In particular, the prediction stage of the algorithm uses the partial differential equations that model the spatio-temporal evolution of image brightness and optical flow to create predictions of such quantities at future times using current estimates. A fast numerical scheme based in finite-difference methods is proposed and

implemented on GPU hardware. The algorithm reaches frame rates in the order of 800 Hz for 640×480 video sequences (4 pix. max. flow) and 500 Hz for 1016×544 image resolution (4 pix. max. flow).

- **Structure flow:** This thesis introduced the structure flow field as the three-dimensional scene flow divided by the depth of the scene. Partial differential equations to model the spatio-temporal evolution of the structure flow field, image brightness, depth and inverse depth were formulated on spherical images. These PDEs can be used both for developing prediction algorithms to compute future values of the fields, or to estimate the structure flow from brightness and depth measurements.

- **Structure flow filter:** An algorithm for the computation of structure flow is proposed. The algorithm follows the same filtering approach as the optical flow filter to develop an algorithm capable of estimating the structure flow in real-time. The algorithm uses image brightness and depth measurements to recover the underlying structure flow. The algorithm is fully implemented on spherical images using the proposed PDEs for structure flow, image brightness and inverse depth.

- **Spherepix data structure:** Chapter 3 introduced the Spherepix data structure for efficient implementation of low-level image operations on spherical images. Spherepix consists of a set of regularized grids on which application of low-level image operators such as Gaussian blurring or gradient computation have the same computational complexity as their implementation on standard perspective images. Spherepix is the numerical back-end used for the real-time implementation of the structure flow algorithm.

5.2 Future Work

The following are future research questions and technological developments that arise from this work.

Research Directions

- **Improve accuracy while maintaining high frame rates:** A particular choice made for the optical and structure flow algorithms was to design the update stage of the algorithms to be as numerically efficient as possible. With this in mind, the update stage of both algorithms was formulated as a per-pixel linear least squares process plus a series of smoothing filters to spread flow estimates to image regions with poor gradient information. While the error analysis performed for both algorithms showed sub-pixel error levels, the estimated flow fields tend to be over-smoothed due to the averaging stage of the algorithm. This can possibly be solved by explicitly imposing smoothing constraints over the flow fields in the update stage of the algorithms. This formulation will lead to a total-variational formulation of the optical and structure flow estimation Horn and Schunck [1981]; Brox et al. [2004]; Herbst et al. [2013]. Total variational methods are in general more accurate than local-based methods, however their computational cost makes them hard to be used on real-time applications. An interesting research question

is the design and development of total-variational flow algorithms capable to work at similar frame rates as those developed in this thesis.

Alternatively, an explicit diffusion term such as proposed in the thesis of Spica Spica [2015] could be considered. An explicit diffusion model in the PDE evolution would have the advantage of stabilising the numerics of the PDE numerical scheme and can be tuned to diffuse most strongly where poor information is available in the image.

- **Edge-preserving numerical prediction schemes:** The numerical schemes implemented for the prediction of both optical and structure flow in the filtering algorithms were based on methods coming from Computational Fluid Dynamics. These methods preserve the overall entropy of the flow fields (Section 2.5). Consequently, the predicted flows suffer of a trailing edge artifact in which background discovered areas blend with foreground objects. While the accuracy of the overall filter algorithms has been demonstrated, this trailing edge artifact will be important to remove in the context of algorithms that need highly accurate state predictions. This cannot be done within the context of physically motivated numerical PDE solvers and it will require the design of new numerical PDE solvers for image processing algorithms.

- **Deployment on real-life robotic platforms:** The quality of both flow filtering algorithms has been illustrated on real-life high-speed video sequences. The next level of research is to use the output of these algorithms for the control of robotic vehicles. One application example is to use structure flow for control of aerial vehicles. It is possible to use the normal component of the structure flow field to control the height of a UAV close to landing. This could improve results of previous works such as Herisse et al. [2012] where the global divergence of the optical flow field was used. Moreover, since the magnitude of structure flow is related to time to colision, it can be used as sensor cue for obstacle avoidance applications. Regions with low magnitude (after derotation) represent safe regions of the scene where the robot can safely move.

- **Visual and inertial sensor fusion:** Chapter 4 developed the PDE equations for the evolution of the structure flow field given the kinematics of the scene and the camera (Equations (4.25), (4.27) and (4.29)). These PDEs assumed the availability of inertial measurements (angular rate and acceleration) to derive external source terms affecting the structure flow field. These inertial measurements can readily be incorporated in the filtering algorithm to improve the accuracy in the prediction stage of the filter. Moreover, it is possible to split the estimated flow into linear and rotational components (Equations (4.15) and (4.16)) and perform analysis on both components separately.

- **Structure flow estimation from omnidirectional camera arrays:** The structure flow equations are formulated on spherical camera geometry. An interesting research problem is the estimation of dense structure flow from omnidirectional camera arrays. An array configuration such as that of Schönbein et al. [2014] can offer dense omnidirectional stereo data to estimate the structure flow. It should be possible to derive PDE equations for the conservation of image brightness in a omnidirectional camera array to estimate structure flow.

- **Multi-camera array image processing on Spherepix:** The Spherepix data structure can be extended to add support for mapping multiple cameras in a single spherical image representation. To achieve this, one can use a multiple camera calibration toolbox Kneip and Li [2014]; Kneip and Furgale [2014] to find the relative pose between cameras, and then place a virtual spherical camera that contains the whole array inside. For each Spherepix pixel, the image value will be taken from all the raw camera images that see in that direction.

Robotic Vision Technologies

- **A real-time robotic vision API:** The performance of real-time vision algorithms, in particular for robotics, depends in great measure on the programmings skills of the person coding the algorithms. While the implementations here presented outperform the state of the art in terms of processing speed, they are ad-hoc software packages difficult to extend.

  An important piece of technology for enabling robotic vision algorithms in real-life applications is a software Application Programming Interface (API) to describe high performance vision algorithms. The API will take a *description* of the algorithm, for example in the form of a compute graph, and perform low-level operations such as kernel scheduling and synchronization. This way, a user of such API can concentrate on creating better algorithms rather than coding low-level real-time code.

  An interesting example of such API is OpenVX from the Khronos Group[1]. OpenVX describes vision algorithms by means of compute graphs. Each node in the graph describes a specific operation (filtering, gradient, Lucas-Kanade optical flow, etc) and each edge entering or leaving the node is a data object (tensor, image, scalar, etc). The compute graph description of the algorithm is portable and platform independent while the implementation of each compute node is device dependent (GPU, FPGA, Accelerator). The availability of a computer vision API standard is compelling, although adoption by the robotics community seems slow.

  Considering that GPUs are currently the most widely used computer vision accelerator, it is interesting to develop a API for prototyping and deployment of real-time vision algorithms on embedded compute platforms. For the development of such API, it is necessary to code compute kernels to be executed by the GPU. There are two frameworks for this: Khronos Group OpenCL[2] and Nvidia CUDA[3]. However, the use of either framework raise several problems. On one hand, Nvidia CUDA is a proprietary framework for Nvidia chips. While the mobile chips made by the company rank in the top in terms of compute performance[4], their market share is limited[5]. On the other hand, although OpenCL standard offers cross-platform support for compute kernels and is supported to

[1] https://www.khronos.org/openvx/
[2] https://www.khronos.org/opencl/
[3] https://developer.nvidia.com/cuda-zone
[4] http://www.notebookcheck.net/Smartphone-Graphics-Cards-Benchmark-List.149363.0.html
[5] http://jonpeddie.com/press-releases/details/qualcomm-single-largest-proprietary-gpu-supplier-imagination-technologies-t/

some extend by mobile chip manufacturers, its adoption, support and application ecosystem is limited in mobile devices[6].

An alternative to both frameworks is to use Khronos Vulkan API[7] as back-end to access the GPU. Vulkan is an hybrid graphics and compute API that offers low-level, low-latency access to the GPU. Vulkan was released early in 2016 and version 1.0 is already supported by all major GPU vendors, both for Desktop systems and mobile devices. Such fast adoption of the standard, primarily coming from the video game industry, is a good indication that Vulkan will be supported and improved over the years to come.

- **A Robotic Embedded Vision device:** This thesis introduced the term *Robotic Embedded Vision* system (REV) to refer to an electronic device containing image sensors and processing capabilities in a single package. Vision algorithms run inside the REV device and the output is sent to the robot's CPU to be used. Currently, REV systems are built using embedded compute boards such as the Nvidia Jetson board[8] or the Odroid XU4[9] plus a camera board and other sensors. These ad-hoc vision platforms make the development of vision algorithms difficult, as programmers need to develop code to control the device in addition to perform the actual computations. The overall performance of the system not only depends on the hardware components used but also on the programming skills of the engineer. This approach makes it hard to transfer vision algorithms from research labs to industry.

  The robotics research and industry communities have leveraged the development of new algorithms and platform thanks to the use of open-hardware and open-source components. Examples of these include the Robot Operating System (ROS)[10] as middleware to access and program robotic platforms and the PixHawk board for controlling UAVs[11]. There is a need in research and industry for REV devices that provide off-the-shelf visual information to robotic platforms. Commercial products, such as the Erle-Brain 2[12] have started to appear in the market.

  A Robotic Embedded Vision system using the real-time computer vision API mentioned above might be an enabling technology for the research, development and deployment of vision algorithms in robotic applications.

[6] http://arrayfire.com/opencl-on-mobile-devices/
[7] https://www.khronos.org/vulkan/
[8] http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html
[9] http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825
[10] http://www.ros.org/
[11] https://pixhawk.org/
[12] http://erlerobotics.com/blog/erle-brain-2/

Appendix A

Ground Truth Optical Flow for a Perspective camera

Consider a point $\mathbf{x} \in \mathbb{R}^3$ in 3D space expressed relative to the camera body fixed frame. Let $\mathbf{v_x}$ denote the velocity of $\mathbf{x}$ expressed in the camera frame. Let $\mathbf{v}_c$ and Ω denote the linear and angular velocity of the camera expressed in the camera frame and let $\Omega_\times$ be the skew-symmetric matrix such that $\Omega \times v = \Omega_\times v$ for any vector v.

The kinemtics of $\mathbf{x}$ are given by

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v_x} - \mathbf{v}_c - \Omega_\times \mathbf{x} \tag{A.1}$$

Next, consider a perspective camera model characterized by the intrinsics matrix $K \in \mathbb{R}^{3\times3}$ and whose focal point is located at the origin of the body fixed frame of the camera. The projection of $\mathbf{x}$ onto the image plane of the camera is

$$\begin{pmatrix}\mathbf{p}\\1\end{pmatrix} = \frac{K\mathbf{x}}{\langle e_3, \mathbf{x}\rangle} \tag{A.2}$$

where $\begin{pmatrix}\mathbf{p}\\1\end{pmatrix}$ is the homogeneous vector representation of pixel $\mathbf{p}$.

The optical flow vector $\mathbf{\Phi} := \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t}$ at pixel $\mathbf{p}$ is calculated as

$$\begin{pmatrix}\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t}\\0\end{pmatrix} = \frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{K\mathbf{x}}{\langle e_3, \mathbf{x}\rangle}\right] \tag{A.3}$$

$$= \frac{\frac{\mathrm{d}}{\mathrm{d}t}\left[K\mathbf{x}\right]\langle e_3, \mathbf{x}\rangle - K\mathbf{x}\frac{\mathrm{d}}{\mathrm{d}t}\left[\langle e_3, \mathbf{x}\rangle\right]}{\langle e_3, \mathbf{x}\rangle^2} \tag{A.4}$$

$$= \frac{K\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}\langle e_3, \mathbf{x}\rangle - K\mathbf{x}\langle e_3, \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}\rangle}{\langle e_3, \mathbf{x}\rangle^2} \tag{A.5}$$

$$= \frac{1}{\langle e_3, \mathbf{x}\rangle}\left[K\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} - \frac{K\mathbf{x}\langle e_3, \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}\rangle}{\langle e_3, \mathbf{x}\rangle}\right] \tag{A.6}$$

$$= \frac{1}{\langle e_3, \mathbf{x}\rangle}\left[K\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} - \begin{pmatrix}\mathbf{p}\\1\end{pmatrix}\langle e_3, \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}\rangle\right] \tag{A.7}$$

Numeric Solution to Spherepix Spring Regularization

B.1 Dynamic System

The position of each mass point is determined by $\boldsymbol{\eta}_{ij}$. Velocity $\dot{\boldsymbol{\eta}}_{ij} \in T_{\eta_{ij}} S^2$ and acceleration $\ddot{\boldsymbol{\eta}}_{ij} \in T_{\eta_{ij}} S^2$ are vector quantities in the tangent space of $\boldsymbol{\eta}_{ij}$. The state can be expressed in beta coordinates as $\boldsymbol{\beta}_{ij} = \mathbf{B}_{ij} \mu_{\text{oth}}(\boldsymbol{\eta}_{ij}; \boldsymbol{\eta}_{ij}) = 0$, $\dot{\boldsymbol{\beta}}_{ij} = \mathbf{B}_{ij} \dot{\boldsymbol{\eta}}_{ij}$ and $\ddot{\boldsymbol{\beta}}_{ij} = \mathbf{B}_{ij} \ddot{\boldsymbol{\eta}}_{ij}$, respectively. The orthonormal basis matrix $\mathbf{B}_{ij} = \begin{pmatrix} \boldsymbol{\nu}_{1,ij}^\top \\ \boldsymbol{\nu}_{2,ij}^\top \end{pmatrix}$ is constructed using $\boldsymbol{\eta}_{ij}$ and $\boldsymbol{\eta}_{i,j+1}$ as

$$\boldsymbol{\nu}_{1,ij} = \frac{\mu_{\text{oth}}(\boldsymbol{\eta}_{i,j+1}; \boldsymbol{\eta}_{ij})}{\| \cdot \|} \tag{B.1}$$

$$\boldsymbol{\nu}_{2,ij} = \boldsymbol{\eta}_{ij} \times \boldsymbol{\nu}_{1,ij} \tag{B.2}$$

$\boldsymbol{\nu}_{ij} \in \mathbb{R}^2$ are the generalized local coordinates of the system expressed in $T_{\eta_{ij}} S^2$. $\delta\boldsymbol{\beta}_{ij} := (\delta\boldsymbol{\beta}_1, \delta\boldsymbol{\beta}_2)$ are the admissible variations of $\boldsymbol{\beta}_{ij}$. Since $\boldsymbol{\beta}_{ij}$ can move on any arbitrary direction in $T_{\eta_{ij}} S^2$, we conclude the system is holonomic.

The Lagrangian at pixel (i, j)

$$\mathcal{L}_{ij} = T_{ij}^* - V_{ij} \tag{B.3}$$

consists of a kinetic coenergy term T_{ij}^* modeling a velocity damper

$$T_{ij}^* = \frac{1}{2} M \dot{\boldsymbol{\beta}}_{ij}^\top \dot{\boldsymbol{\beta}}_{ij} \tag{B.4}$$

and a potential energy term V_{ij} considering the elongation of all springs attached to mass point $\boldsymbol{\eta}_{ij}$

$$V_{ij} = \frac{1}{2} \sum_{r,c \in \Omega_{ij}} K(\|\boldsymbol{\beta}_{ij} - \boldsymbol{\beta}_{rc}\| - L_{ij})^2 \tag{B.5}$$

where Ω_{ij} denotes the 8-neighborhood of $\boldsymbol{\eta}_{ij}$. L_{ij} is the rest elongation of spring connecting $\boldsymbol{\eta}_{ij}$ and $\boldsymbol{\eta}_{rc}$. The value is set to L for springs connecting in either row or column direction, and $\sqrt{2}L$ for springs in the diagonal direction. The Lagrange equation describing the dynamic

system is

$$\frac{\partial}{\partial t}\left(\frac{\partial \mathcal{L}_{ij}}{\partial \dot{\boldsymbol{\beta}}_{ij}}\right) - \frac{\partial \mathcal{L}_{ij}}{\partial \boldsymbol{\beta}_{ij}} = F_{ij} \tag{B.6}$$

After some algebraic manipulation, we obtain an ordinary differential equation for state $(\boldsymbol{\eta}_{ij}, \dot{\boldsymbol{\eta}}_{ij}, \ddot{\boldsymbol{\eta}}_{ij})$ represented in beta coordinates

$$M\ddot{\boldsymbol{\beta}}_{ij} + \sum_{r,c\in\Omega_{ij}} K(\|\boldsymbol{\beta}_{rc}\| - L_{rc})\frac{(-\boldsymbol{\beta}_{rc})}{\|\boldsymbol{\beta}_{rc}\|} = -c\dot{\boldsymbol{\beta}}_{ij} \tag{B.7}$$

B.2 Numerical solution

We use an Euler integration scheme to solve the Lagrange equation (B.7). State $(\boldsymbol{\eta}^n_{ij}, \dot{\boldsymbol{\eta}}^n_{ij}, \ddot{\boldsymbol{\eta}}^n_{ij})$, where n denotes time step index, is transformed to its equivalent beta coordinates version as

$$\boldsymbol{\beta}^n_{ij} = \mathbf{B}^n_{ij}\mu_{\text{oth}}(\boldsymbol{\eta}^n_{ij}; \boldsymbol{\eta}^n_{ij}) = 0 \tag{B.8}$$

$$\dot{\boldsymbol{\beta}}^n_{ij} = \mathbf{B}^n_{ij}\dot{\boldsymbol{\eta}}^n_{ij} \tag{B.9}$$

$$\ddot{\boldsymbol{\beta}}^n_{ij} = \mathbf{B}^n_{ij}\ddot{\boldsymbol{\eta}}^n_{ij} \tag{B.10}$$

Let Δt by the time step integration. The Euler forward integration of Equation (B.7) follows

$$\ddot{\boldsymbol{\beta}}^{n+1}_{ij} = \frac{1}{M}\left(-c\dot{\boldsymbol{\beta}}^n_{ij} + \sum_{r,c\in\Omega_{ij}} K(\|\boldsymbol{\beta}^n_{rc}\| - L_{rc})\frac{\boldsymbol{\beta}^n_{rc}}{\|\cdot\|}\right) \tag{B.11}$$

$$\dot{\boldsymbol{\beta}}^{n+1}_{ij} = \dot{\boldsymbol{\beta}}^n_{ij} + \Delta t\ddot{\boldsymbol{\beta}}^{n+1}_{ij} \tag{B.12}$$

$$\boldsymbol{\beta}^{n+1}_{ij} = \Delta t\dot{\boldsymbol{\beta}}^{n+1}_{ij} \tag{B.13}$$

After one time step, state variables are expressed in their original coordinate system. First, $\boldsymbol{\mu}^{n+1}_{ij} = B^{n\top}_{ij}\boldsymbol{\beta}^{n+1}_{ij}$ is the position expressed in tangent space coordinates. The new position in spherical coordinates is

$$\boldsymbol{\eta}^{n+1}_{ij} = \mu^{-1}_{\text{oth}}(\boldsymbol{\mu}^{n+1}_{ij}; \boldsymbol{\eta}^n_{ij}) \tag{B.14}$$

For the velocity and acceleration vectors, we rotate their new values $\dot{\boldsymbol{\eta}}^*_{ij} = B^{n\top}_{ij}\dot{\boldsymbol{\beta}}^{n+1}_{ij}$ and $\ddot{\boldsymbol{\eta}}^*_{ij} = B^{n\top}_{ij}\ddot{\boldsymbol{\beta}}^{n+1}_{ij}$ by a rotation matrix R created from axis $\mathbf{r} = \boldsymbol{\eta}^n_{ij} \times \boldsymbol{\eta}^{n+1}_{ij}/\|\cdot\|$ and angle $\theta = \arccos(\langle\boldsymbol{\eta}^n_{ij}, \boldsymbol{\eta}^{n+1}_{ij}\rangle)$

$$\dot{\boldsymbol{\eta}}^{n+1}_{ij} = R^{n+1}_{ij}\dot{\boldsymbol{\eta}}^*_{ij} \tag{B.15}$$

$$\ddot{\boldsymbol{\eta}}^{n+1}_{ij} = R^{n+1}_{ij}\ddot{\boldsymbol{\eta}}^*_{ij} \tag{B.16}$$

The process is repeated from Equation (B.8) to Equation (B.16) for a fixed number of iterations or until the velocity $\dot{\boldsymbol{\eta}}^n_{ij}$ for all points in the face fall bellow a certain threshold.

Bibliography

ABSIL, P.-A.; MAHONY, R.; AND SEPULCHRE, R., 2009. *Optimization algorithms on matrix manifolds*. Princeton University Press. (cited on page 51)

ADARVE, J. D. AND MAHONY, R., 2016a. A filter formulation for computing real time optical flow. *IEEE Robotics and Automation Letters*, 1, 2 (July 2016), 1192–1199. (cited on page 15)

ADARVE, J. D. AND MAHONY, R., 2016b. Real-time structure flow. *IEEE Transactions on Robotics*, (2016). (under review). (cited on pages 1 and 72)

ADARVE, J. D. AND MAHONY, R., 2017. Spherepix: A data structure for spherical image processing. *IEEE Robotics and Automation Letters*, 2, 2 (April 2017), 483–490. (cited on page 46)

ANANDAN, P., 1989. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2 (1989), 283–310. (cited on page 11)

ANGUITA, M.; DIAZ, J.; ROS, E.; AND FERNANDEZ-BALDOMERO, F. J., 2009. Optimization strategies for high-performance computing of optical-flow in general-purpose processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 19, 10 (Oct 2009), 1475–1488. (cited on page 15)

BAGNATO, L.; FROSSARD, P.; AND VANDERGHEYNST, P., 2009. Optical flow and depth from motion for omnidirectional images using a tv-l1 variational framework on graphs. In *Proc. IEEE Int. Conf. Img. Proc.*, 1469–1472. (cited on page 48)

BAKER, S.; SCHARSTEIN, D.; LEWIS, J.; ROTH, S.; BLACK, M.; AND SZELISKI, R., 2011. A database and evaluation methodology for optical flow. *Int. J. Comput. Vision*, 92 (2011), 1–31. (cited on pages 3, 10, 11, 31, 32, 42, and 92)

BANGURA, M.; KUIPERS, F.; ALLIBERT, G.; AND MAHONY, R., 2015. Non-linear velocity aided attitude estimation and velocity control for quadrotors. In *16th Australasian Conference on Robotics and Automation*. (cited on pages 4 and 9)

BAO, L.; YANG, Q.; AND JIN, H., 2014. Fast edge-preserving patchmatch for large displacement optical flow. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, 3534–3541. (cited on page 11)

BARRANCO, F.; TOMASI, M.; DÍAZ, J.; VANEGAS, M.; AND ROS, E., 2013. Pipelined architecture for real-time cost-optimized extraction of visual primitives based on FPGAs. *Digital Signal Processing*, 23, 2 (2013), 675 – 688. (cited on page 15)

BARRON, J., 1994. System and experiment. performance of optical flow techniques. *Int. J. Comput. Vision*, 12:1 (1994), 43–77. (cited on pages 1, 3, 10, 12, 67, 72, and 82)

BAUMGARDNER, J. AND FREDERICKSON, P., 1985. Icosahedral discretization of the two-sphere. *SIAM Journal on Numerical Analysis*, 22, 6 (1985), 1107–1115. (cited on page 47)

BENOSMAN, R.; CLERCQ, C.; LAGORCE, X.; IENG, S.-H.; AND BARTOLOZZI, C., 2014. Event-based visual flow. *Neural Networks and Learning Systems, IEEE Transactions on*, 25, 2 (Feb 2014), 407–417. (cited on page 16)

BENOSMAN, R.; IENG, S.-H.; CLERCQ, C.; BARTOLOZZI, C.; AND SRINIVASAN, M., 2012. Asynchronous frameless event-based optical flow. *Neural Networks*, 27 (2012), 32 – 37. (cited on page 16)

BLACK, M. J., 1992. *Robust incremental optical flow*. Ph.D. thesis, Yale university. (cited on page 16)

BONNABEL, S. AND ROUCHON, P., 2009. Fusion of inertial and visual: a geometrical observer-based approach. In *Proc. Mediterranean Conf. on Intelligent Systems and Automation*, vol. 1107, 54. (cited on page 83)

BORST, A., 2009. Drosophila's view on insect vision. *Current Biology*, 19, 1 (2009), R36 – R47. (cited on page 15)

BOUGUET, J.-Y., 2001. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. Technical report, Intel Corporation. (cited on pages 4, 13, 14, 15, and 31)

BROX, T.; BRHUN, A.; PAPENBERG, N.; AND WEICKERT, J., 2004. High accuracy optical flow estimation based on a theory for warping. In *Proc. European Conf. Comput. Vision*, vol. 4, 25–36. (cited on pages 11, 14, 31, 68, 77, and 100)

BRUHN, A.; WEICKERT, J.; AND SCHN?RR, C., 2005. Lucas kanade meets horn schunck: Combining local and global optic flow methods. *International Journal of Computer Vision*, 61 (2005), 211–231. (cited on page 11)

BUTLER, D. J.; WULFF, J.; STANLEY, G. B.; AND BLACK, M. J., 2012. A naturalistic open source movie for optical flow evaluation. In *Proc. European Conf. Comput. Vision*, 611–625. (cited on pages 3, 10, 11, and 31)

CHAHL, J. S. AND SRINIVASAN, M. V., 2000. Filtering and processing of panoramic images obtained using a camera and a wide-angle-imaging reflective surface. *J. Opt. Soc. Am. A*, 17, 7 (Jul 2000), 1172–1176. (cited on page 45)

CHAO, H.; GU, Y.; AND NAPOLITANO, M., 2014. A survey of optical flow techniques for robotics navigation applications. *Journal of Intelligent & Robotic Systems*, 73, 1-4 (2014), 361–372. (cited on page 12)

CRUZ-MOTA, J.; BOGDANOVA, I.; PAQUIER, B.; BIERLAIRE, M.; AND THIRAN, J.-P., 2012. Scale invariant feature transform on the sphere: Theory and applications. *Int. J. Comput. Vision*, 98, 2 (2012), 217–241. (cited on pages 49, 64, and 65)

DANIILIDIS, K.; MAKADIA, A.; AND BULOW, T., 2002. Image processing in catadioptric planes: spatiotemporal derivatives and optical flow computation. In *Proc. 3rd Workshop Omnidirectional Vision*, 3–10. (cited on page 48)

DEMONCEAUX, C.; VASSEUR, P.; AND FOUGEROLLE, Y., 2011. Central catadioptric image processing with geodesic metric. *Image and Vision Computing*, 29, 12 (2011), 840 – 849. (cited on page 46)

DEROME, M.; PLYER, A.; SANFOURCHE, M.; AND LE BESNERAIS, G., 2016. *A Prediction-Correction Approach for Real-Time Optical Flow Computation Using Stereo*, 365–376. Springer International Publishing, Cham. ISBN 978-3-319-45886-1. (cited on page 15)

FARNEBÄCK, G., 2003. Two-frame motion estimation based on polynomial expansion. In *Image Analysis*, 363–370. ISBN 978-3-540-40601-3. (cited on pages 15 and 31)

GEIGER, A.; LENZ, P.; STILLER, C.; AND URTASUN, R., 2013. Vision meets robotics: The kitti dataset. *Int. J. Robotics Reseach*, (Aug. 2013). (cited on pages 3, 10, 11, and 31)

GEYER, C. AND DANIILIDIS, K., 2001. Catadioptric projective geometry. *International Journal of Computer Vision*, 45, 3 (2001), 223–243. (cited on page 45)

GONG, M., 2009. Real-time joint disparity and disparity flow estimation on programmable graphics hardware. *Comput. Vision and Image Understanding*, 113, 1 (2009), 90 – 100. (cited on page 77)

GONG, M. AND YANG, Y.-H., 2006. Disparity flow estimation using orthogonal reliability-based dynamic programming. In *Proc. IEEE Int. Conf. Pattern Recognition*, vol. 2, 70–73. (cited on page 76)

GÓRSKI, K. M.; HIVON, E.; BANDAY, A. J.; WANDELT, B. D.; HANSEN, F. K.; REINECKE, M.; AND BARTELMANN, M., 2005. Healpix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622, 2 (2005), 759. (cited on pages 47 and 49)

HADFIELD, S. AND BOWDEN, R., 2011. Kinecting the dots: Particle based scene flow from depth sensors. In *Proc. IEEE Conf. Comput. Vision*, 2290–2295. (cited on pages 3 and 76)

HAMEL, T. AND MAHONY, R., 2002. Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach. *IEEE Transactions on Robotics and Automation*, 18, 2 (Apr 2002), 187–198. (cited on pages 3, 9, 72, and 81)

HANSEN, P.; CORKE, P.; AND BOLES, W., 2010. Wide-angle visual feature matching for outdoor localization. *Int. J. Robotics Research*, 29, 2-3 (2010), 267–297. (cited on pages 50 and 65)

HARA, K.; INOUE, K.; AND URAHAMA, K., 2015. Gradient operators for feature extraction from omnidirectional panoramic images. *Pattern Recognition Letters*, 54 (2015), 89 – 96. (cited on page 48)

HERBST, E.; REN, X.; AND FOX, D., 2013. Rgb-d flow: Dense 3-d motion estimation using color and depth. In *Proc. IEEE Int. Conf. Robotics and Automation*, 2276–2282. (cited on pages 3, 77, and 100)

HERISSE, B.; HAMEL, T.; MAHONY, R.; AND RUSSOTTO, F.-X., 2012. Landing a vtol unmanned aerial vehicle on a moving platform using optical flow. *IEEE. Trans. Robotics*, 28, 1 (2012), 77–89. (cited on pages 3, 9, 15, and 101)

HERISSE, B.; RUSSOTTO, F.-X.; HAMEL, T.; AND MAHONY, R., 2008. Hovering flight and vertical landing control of a vtol unmanned aerial vehicle using optical flow. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 801–806. (cited on page 72)

HORN, B. K. AND SCHUNCK, B. G., 1981. Determining optical flow. *Artificial Intelligence*, 17, 1 (1981), 185 – 203. (cited on pages 10, 11, 13, and 100)

HUGUET, F. AND DEVERNAY, F., 2007. A variational method for scene flow estimation from stereo sequences. In *Proc. IEEE Conf. Comput. Vision*, 1–7. (cited on page 76)

KAGEYAMA, A. AND SATO, T., 2004. âĂIJyin-yang gridâĂİ: An overset grid in spherical geometry. *Geochemistry, Geophysics, Geosystems*, 5, 9 (2004). (cited on pages 48 and 50)

KNEIP, L. AND FURGALE, P., 2014. Opengv: A unified and generalized approach to real-time calibrated geometric vision. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 1–8. (cited on page 102)

KNEIP, L. AND LI, H., 2014. Efficient computation of relative pose for multi-camera systems. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (cited on page 102)

KOENDERINK, J. J. AND VAN DOORN, A. J., 1987. Facts on optic flow. *Biological Cybernetics*, 56, 4 (1987), 247–254. (cited on page 81)

KROEGER, T.; TIMOFTE, R.; DAI, D.; AND VAN GOOL, L., 2016. Fast optical flow using dense inverse search. *arXiv preprint arXiv:1603.03590*, (2016). (cited on page 15)

LETOUZEY, A.; PETIT, B.; AND BOYER, E., 2011. Surface flow from depth and color images. In *British Machine Vision Conference*, 46–1. (cited on page 76)

LEVEQUE, R. J., 1992. *Numerical methods for conservation laws*, vol. 132. Springer. (cited on page 23)

LEVEQUE, R. J., 2002. *Finite volume methods for hyperbolic problems*, vol. 31. Cambridge university press. (cited on pages 20 and 23)

LOWE, D., 2004. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60, 2 (2004), 91–110. (cited on page 63)

LUCAS, B. D. AND KANADE, T., 1981. An iterative image registration technique with an application to stereo vision. In *Proc. 7th Int. Joint Conf. Artif. Intell.*, 674–679. Vancouver, BC, Canada. (cited on pages 1, 10, 11, 13, and 14)

MAFRICA, S.; SERVEL, A.; AND RUFFIER, F., 2016. Optic-flow based car-like robot operating in a 5-decade light level range. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 5568–5575. (cited on page 15)

MCCARTHY, C. AND BARNES, N., 2012. A unified strategy for landing and docking using spherical flow divergence. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34, 5 (May 2012), 1024–1031. (cited on page 72)

MENZE, M. AND GEIGER, A., 2015. Object scene flow for autonomous vehicles. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (cited on pages 3 and 76)

MUEGGLER, E.; HUBER, B.; AND SCARAMUZZA, D., 2014. Event-based, 6-dof pose tracking for high-speed maneuvers. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2761–2768. (cited on pages 14 and 16)

NELSON, R. C. AND ALOIMONOS, J., 1989. Obstacle avoidance using flow field divergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11, 10 (1989), 1102–1106. (cited on page 72)

PATRAS, I.; ALVERTOS, N.; AND TZIRITAS, G., 1996. Joint disparity and motion field estimation in stereoscopic image sequences. In *Proc. IEEE Int. Conf. on Pattern Recognition*, vol. 1, 359–363 vol.1. (cited on page 76)

PLETT, J.; BAHL, A.; BUSS, M.; K?HNLENZ, K.; AND BORST, A., 2012. Bio-inspired visual ego-rotation sensor for mavs. *Biological Cybernetics*, 106 (2012), 51–63. (cited on page 15)

PLYER, A.; LE BESNERAIS, G.; AND CHAMPAGNAT, F., 2014. Massively parallel lucas kanade optical flow for real-time video processing applications. *J. Real-Time Img. Processing*, (2014), 1–18. (cited on pages 14, 15, 31, 34, and 40)

PUIG, L.; GUERRERO, J.; AND DANIILIDIS, K., 2014. Scale space for camera invariant features. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36, 9 (Sept 2014), 1832–1846. (cited on pages 50 and 65)

PUTMAN, W. M. AND LIN, S.-J., 2007. Finite-volume transport on various cubed-sphere grids. *J. Computational Physics*, 227, 1 (2007), 55 – 78. (cited on pages 48 and 54)

QUIROGA, J.; DEVERNAY, F.; AND CROWLEY, J., 2012. Scene flow by tracking in intensity and depth data. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognition Workshops*, 50–57. (cited on page 76)

RABE, C.; MÜLLER, T.; WEDEL, A.; AND FRANKE, U., 2010. Dense, robust, and accurate motion field estimation from stereo image sequences in real-time. In *Proc. European Conf. Comput. Vision*, 582–595. (cited on pages 3 and 77)

REICHARDT, W., 1987. Evaluation of optical motion information by movement detectors. *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, 161 (1987), 533–547. (cited on page 15)

RONCHI, C.; IACONO, R.; AND PAOLUCCI, P., 1996. The "cubed sphere": A new method for the solution of partial differential equations in spherical geometry. *J. Computational Physics*, 124, 1 (1996), 93 – 114. (cited on pages 48, 49, and 54)

RUFFIER, F. AND FRANCESCHINI, N., 2005. Optic flow regulation: the key to aircraft automatic guidance. *Robotics and Autonomous Systems*, 50, 4 (2005), 177 – 194. (cited on pages 3, 9, 15, and 72)

SCARAMUZZA, D.; MARTINELLI, A.; AND SIEGWART, R., 2006. A toolbox for easily calibrating omnidirectional cameras. In *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Sys.*, 5695–5701. (cited on pages 46, 50, 59, 61, and 64)

SCHÖNBEIN, M.; STRAUS, T.; AND GEIGER, A., 2014. Calibrating and centering quasi-central catadioptric cameras. In *Proc. IEEE Conf. Robot. Autom.*, 4443–4450. (cited on pages 45, 46, 50, 68, and 101)

SPICA, R., 2015. *Contributions to active visual estimation and control of robotic systems*. Ph.D. thesis, Université de Rennes 1. (cited on page 101)

SPIES, H.; JÃĎHNE, B.; AND BARRON, J. L., 2002. Range flow estimation. *Comput. Vision and Image Understanding*, 85, 3 (2002), 209 – 231. doi:http://dx.doi.org/10.1006/cviu.2002.0970. (cited on pages 76 and 89)

SRINIVASAN, M. V., 1994. An image interpolation technique for the computation of optical flow and egomotion. *Biological Cybernetics*, 71 (1994), 401–415. (cited on pages 4, 11, and 14)

SRINIVASAN, M. V., 2011a. Honeybees as a model for the study of visually guided flight, navigation, and biologically inspired robotics. *Physiological Review*, 91 (2011), 389–411. (cited on pages 3, 9, 11, and 15)

SRINIVASAN, M. V., 2011b. Visual control of navigation in insects and its relevance for robotics. *Current Opinion in Neurobiology*, 21, 4 (2011), 535 – 543. (cited on pages 15 and 72)

SRINIVASAN, M. V.; ZHANG, S. W.; CHAHL, J. S.; BARTH, E.; AND VENKATESH, S., 2000. How honeybees make grazing landings on flat surfaces. *Biological Cybernetics*, 83, 3 (2000), 171–183. (cited on page 72)

SUN, D.; ROTH, S.; AND BLACK, M. J., 2014. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106, 2 (2014), 115–137. (cited on page 12)

TAO, M.; BAI, J.; KOHLI, P.; AND PARIS, S., 2012. Simpleflow: A non-iterative, sublinear optical flow algorithm. *Comput. Graphics Forum*, 31, 2pt1 (2012), 345–353. (cited on page 11)

THOMAS, J. W., 1995. *Numerical partial differential equations: finite difference methods*, vol. 22. Springer. (cited on pages 10, 23, 27, and 90)

THOMAS, J. W., 1999. *Numerical partial differential equations: conservation laws and elliptic equations*, vol. 3. Springer Berlin. (cited on pages 23, 25, and 90)

VEDULA, S.; BAKER, S.; RANDER, P.; COLLINS, R.; AND KANADE, T., 1999. Three-dimensional scene flow. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 722–729 vol.2. (cited on pages 1, 72, 75, and 78)

VEDULA, S.; RANDER, P.; COLLINS, R.; AND KANADE, T., 2005. Three-dimensional scene flow. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27, 3 (March 2005), 475–480. (cited on page 75)

VOGEL, C.; SCHINDLER, K.; AND ROTH, S., 2015. 3d scene flow estimation with a piecewise rigid scene model. *Int. J. Comput. Vision*, 115, 1 (2015), 1–28. (cited on page 76)

WEDEL, A.; BROX, T.; VAUDREY, T.; RABE, C.; FRANKE, U.; AND CREMERS, D., 2011. Stereoscopic scene flow computation for 3d motion understanding. *Int. J. Comput. Vision*, 95, 1 (2011), 29–51. (cited on pages 3, 76, and 77)

WERLBERGER, M., 2012. *Convex Approaches for High Performance Video Processing*. Ph.D. thesis, Institute for Computer Graphics and Vision, Graz University of Technology. (cited on pages 11 and 31)

YAMAMOTO, M.; BOULANGER, P.; BERALDIN, J. A.; AND RIOUX, M., 1993. Direct estimation of range flow on deformable shape from a video rate range camera. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15, 1 (1993), 82–89. (cited on page 76)

YANG, J. AND LI, H., 2015. Dense, accurate optical flow estimation with piecewise parametric model. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (cited on page 11)

ZABIH, R. AND WOODFILL, J., 1994. *Non-parametric local transforms for computing visual correspondence*, 151–158. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-48400-4. (cited on page 14)

ZARROUATI, N.; ALDEA, E.; AND ROUCHON, P., 2012. So(3)-invariant asymptotic observers for dense depth field estimation based on visual data and known camera motion. In *American Control Conference*, 4116–4123. (cited on page 83)

ZHANG, T.; WU, H.; BORST, A.; KUHNLENZ, K.; AND BUSS, M., 2008. An fpga implementation of insect-inspired motion detector for high-speed vision systems. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 335 –340. (cited on page 15)

ZHANG, Z.; REBECQ, H.; FORSTER, C.; AND SCARAMUZZA, D., 2016. Benefit of large field-of-view cameras for visual odometry. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 801–808. (cited on pages 3, 64, and 92)